Практическое программирование

Python на ЕГЭ

справочное учебное пособие по подготовке
к ЕГЭ по информатике

В.Ю.Поликарпов, Ю.Н.Поликарпов

самостоятельное учебное электронное издание
предназначено для использования
в процессе подготовки к ЕГЭ по информатике

СПб
Издатель Ю.Н.Поликарпов
2026

ISBN 978-5-6050573-4-5

© Ю.Н.Поликарпов, 2026

УДК 372.862
ББК 74.263.2
П50

П50 **Поликарпов, В.Ю., Поликарпов, Ю.Н.** Практическое программирование. Python на ЕГЭ : справочное учебное пособие по подготовке к ЕГЭ по информатике : самостоятельное учебное электронное издание / В.Ю.Поликарпов, Ю.Н.Поликарпов. — 1-е издание. — СПб. : издатель Ю.Н.Поликарпов, 2026. — 1 CD-R (2 Мб). — ISBN 978-5-6050573-4-5. — Текст : электронный.

Справочное учебное пособие содержит курс подготовки к ЕГЭ по информатике. В пособии кратко изложены основные численные методы и их реализация на Python, которые можно использовать для решения типовых задач ЕГЭ.

В первую очередь пособие адресовано школьникам и учителям, осуществляющим подготовку к экзамену по информатике.

Пособие распространяется под открытой лицензией на условиях использования для личных некоммерческих учебных целей при соблюдении всех трех условий одновременно. Использование для иных целей с письменного разрешения авторов.

Минимальные системные требования:

Процессор Intel® или AMD с частотой не менее 1,5 ГГц;
Windows 11, Windows 10 версии 1809 или более поздней, Windows Server 2016 или Windows Server 2019; 4 ГБ оперативной памяти (16 ГБ желательно); 20 ГБ свободного пространства на жестком диске; разрешение экрана 1024x768; аппаратное ускорение видеоплаты (желательно); Adobe Acrobat Standard.

Издательская система LaTeX.
Компьютерная верстка, редактура, корректура Ю.Н.Поликарпова.

Дата подписания к использованию 15.07.2026.
Объем издания 2 МБ.
Комплектация 1 CD-R, пластиковый бокс.
Тираж 100 экз.
Издатель Ю.Н.Поликапов, newwave50@bk.ru, www.dec.su, +79516739548.

Оглавление

Введение

О курсе

Этот курс — полная система подготовки к ЕГЭ по информатике с нуля до 100 баллов. Мы пройдём все 27 заданий экзамена, освоим Python, Excel и ручные методы решения.

Цели курса

- Освоить Python на уровне, достаточном для решения 18 заданий ЕГЭ.
- Научиться применять Excel / LibreOffice для заданий №3, 9, 18.
- Освоить ручные методы для заданий №1, 4, 7, 10, 11, 13, 15.
- Выработать стратегию экзамена: порядок, тайминг, чек-листы.

Структура курса (9 разделов, 33 занятия)

Раздел	Заня-тия	Содержание
0. Инструменты	1	Структура ЕГЭ, установка Python, Excel
1. База Python	2–7	Синтаксис, типы, циклы, строки, списки, функции
2. Брутфорс	8–13	`itertools`, перебор, таблицы истинности, СС, логика, делители
3. СС и строки	14–15	Системы счисления, редактор строк, скользящее окно
4. Рекурсия	16–20	Рекурсия, мемоизация, рекурсивный перебор, подсчёт путей, параллельные процессы
5. Теория игр	21–23	`game()`, `any/all`, №19–21
6. Динамическое программи-рование	24–27	Нисходящее/восходящее ДП, цифровое ДП, ДП по остаткам, кла-стеризация
7. Файлы и данные	28–29	Чтение файлов, сортировка, жадные алгоритмы
8. Excel и ручные задачи	30–32	Графы, сети, логика, ДП в таблице, кодирование, память, поиск, пароли
9. Стратегия экзамена	33	Порядок, тайминг, ошибки, чек-лист

Инструменты

Для прохождения курса вам понадобятся:

- **Python 3.9+** — основной инструмент. Установите с python.org.
- **Среда разработки (IDE):** PyCharm Community, VS Code или IDLE (встроенный).
- **Jupyter Notebook** — для интерактивных занятий (установка: `pip install notebook`).
- **LibreOffice Calc** (или Excel) — для заданий №3, 9, 18.
- **Текстовый редактор** (Блокнот, Notepad++) — для задания №10.
- **Калькулятор** — для расчётов объёма памяти, информационного объёма.

Как работать с курсом

1. **Последовательно.** Занятия строятся друг на друге. Не перескакивайте.
2. **Практика.** После каждого занятия — домашнее задание. Выполняйте его.
3. **Шаблоны.** Копируйте шаблоны кода в отдельный файл — они пригодятся на экзамене.
4. **Пробники.** После модуля 8 начните решать полные варианты.
5. **Чек-листы.** Используйте чек-лист из занятия 33 для контроля готовности.

Начните с первого занятия

Откройте **Занятие 1. Инструменты** и начните подготовку.

Удачи!

Глава 0. ИНСТРУМЕНТЫ

Занятие 1. Инструменты

План занятия

1. Структура ЕГЭ: 27 заданий, баллы, распределение
2. Три инструмента: Python, Excel, ручной счёт
3. Установка и настройка Python
4. Первая программа на Python
5. Excel: интерфейс и простые формулы
6. Текстовый редактор: поиск и замена

> **Цель занятия:** Ознакомиться с инструментами, используемыми на экзамене.

1. Структура ЕГЭ по информатике

Общая информация

- **Всего заданий:** 27
- **Время экзамена:** 3 часа 55 минут (235 минут)
- **Максимальный первичный балл:** 29
- **Минимальный порог:** 6 первичных баллов (40 тестовых)

Распределение заданий по сложности

Уровень	Задания	Баллы
Базовый	№1–10	10 баллов (по 1)
Повышенный	№11–23	13 баллов (по 1)
Высокий	№24–27	6 баллов (№24,25 — по 1; №26,27 — по 2)

Перевод первичных баллов в тестовые (примерный)

Первичный	Тестовый
6	40
10	50
15	62
20	74
25	88
29	100

> **Совет:** Для поступления в хороший вуз нужно 80+ тестовых баллов. Это примерно 20–22 первичных балла.

2. Три инструмента для решения задач

Python (18 заданий)

Python — главный инструмент. На нём решаем задачи, где нужен перебор, вычисления, анализ данных.

Метод	Задания
Брутфорс (перебор)	№2, 5, 8, 14, 15, 25
Рекурсия + кэш	№16, 19, 20, 21, 22, 23
Динамическое программирование	№8(сл), 16(сл), 23(сл), 27
Строки и файлы	№12, 17, 24
Сортировка и жадность	№26

Excel и Calc (4 задания)

Excel быстрее для задач с таблицами и фильтрацией.

Задание	Суть
№3	Базы данных: фильтрация, сортировка
№9	Электронные таблицы: ВПР, СЧЁТЕСЛИ
№18	Логика + таблицы

Ручной счёт / калькулятор (8 заданий)

Некоторые задачи быстрее решить руками, чем писать код.

Задание	Суть
№1	Кратчайший путь в графе
№4	Кодирование, условие Фано
№7	Объём памяти (изображения, звук)
№10	Текстовый поиск (Ctrl+F)
№11	Информационный объём, пароли
№13	Количество путей в графе
№15	Логика с отрезками (сложные)

> **Ключевой принцип:** Не писать код там, где быстрее посчитать руками или в Excel.

3. Установка и настройка Python

Что нужно установить

1. **Python** — с официального сайта python.org
   - Версия: 3.9 или выше
   - **Важно:** при установке отметить галочку «Add Python to PATH»
2. **IDE (среда разработки):**
   - **PyCharm Community** — бесплатная, мощная
   - **VS Code** — легковесная, с расширениями
   - **IDLE** — встроенная в Python (минимальная)
3. **Jupyter Notebook** — для интерактивных занятий

```
pip install notebook
```

Запуск:

```
jupyter notebook
```

Проверка установки

Откройте терминал (командную строку) и выполните:

```
python --version
```

Или внутри Python:

```
import sys
print(f"Версия Python: {sys.version}")
print(f"Исполняемый файл: {sys.executable}")
```

Библиотеки, которые понадобятся

На ЕГЭ доступны **только стандартные библиотеки** Python. Вот те, что мы будем использовать:

Библиотека	Для чего
`itertools`	Перебор комбинаций (product, permutations)
`functools`	Кэширование (`@lru_cache`)
`math`	Математические функции (gcd, sqrt, isqrt)
`collections`	Структуры данных (Counter, deque)
`sys`	Системные функции (setrecursionlimit)

Проверка доступности:

```
import itertools
import functools
import math
```

```python
import collections
import sys

print("   Все необходимые библиотеки доступны")
```

4. Первая программа на Python

«Hello, World!» и базовый ввод/вывод

```python
# Самая простая программа
print("Hello, World!")

# Ввод и вывод
name = input("Введите ваше имя: ")
print(f"Привет, {name}! Добро пожаловать на курс.")
```

Простые вычисления

```python
# Арифметика
a = 10
b = 3

print(f"Сложение:    {a} + {b} = {a + b}")
print(f"Вычитание:   {a} - {b} = {a - b}")
print(f"Умножение:   {a} * {b} = {a * b}")
print(f"Деление:     {a} / {b} = {a / b}")
print(f"Целочисл.:   {a} // {b} = {a // b}")
print(f"Остаток:     {a} % {b} = {a % b}")
print(f"Степень:     {a} ** {b} = {a ** b}")

# Типы данных
x = 42                 # int — целое число
y = 3.14               # float — дробное
s = "ЕГЭ"              # str — строка
flag = True            # bool — логическое

print(f"{x} — {type(x)}")
print(f"{y} — {type(y)}")
print(f"{s} — {type(s)}")
print(f"{flag} — {type(flag)}")

# Преобразование типов
num_str = "123"
num_int = int(num_str)     # строка → число
num_float = float(num_str)  # строка → дробное

print(f"Строка:  {num_str} — {type(num_str)}")
print(f"Число:   {num_int} — {type(num_int)}")
print(f"Дробное: {num_float} — {type(num_float)}")

# Обратное преобразование
back_to_str = str(num_int)
print(f"Снова строка: {back_to_str} — {type(back_to_str)}")
```

Мини-задача

Напишите программу, которая запрашивает два числа и выводит их сумму, разность и произведение.

```
# Решение мини-задачи
a = int(input("Введите первое число: "))
b = int(input("Введите второе число: "))

print(f"{a} + {b} = {a + b}")
print(f"{a} - {b} = {a - b}")
print(f"{a} * {b} = {a * b}")
```

5. Excel и Calc: интерфейс и простые формулы

Что нужно уметь в Excel/Calc

Для ЕГЭ достаточно базовых навыков:

Действие	Как сделать
Фильтрация	Данные → Фильтр → выбрать значения
Сортировка	Данные → Сортировка → по столбцу
Автосумма	Выделить ячейки → Σ (Alt + =)
СЧЁТЕСЛИ	`=СЧЁТЕСЛИ(диапазон; критерий)`
СУММЕСЛИ	`=СУММЕСЛИ(диапазон; критерий; диапазон_суммирования)`
ЕСЛИ	`=ЕСЛИ(условие; значение_если_истина; значение_если_ложь)`
ВПР	`=ВПР(искомое; таблица; столбец; ЛОЖЬ)`

Простые формулы

```
=A1 + B1          — сложение
=A1 * B1          — умножение
=СУММ(A1:A10)     — сумма диапазона
=СРЗНАЧ(A1:A10)   — среднее арифметическое
=МИН(A1:A10)      — минимум
=МАКС(A1:A10)     — максимум
```

Практическое задание (открыть в Excel/Calc)

Создайте таблицу:

Товар	Цена	Количество	Сумма
Ручка	30	5	=B2*C2
Тетрадь	45	10	=B3*C3
Карандаш	15	20	=B4*C4
Ластик	25	8	=B5*C5

1. В столбце D вычислите сумму (Цена × Количество)

2. В ячейке D6 посчитайте **общую сумму** через `=СУММ(D2:D5)`

3. Добавьте фильтр и отсортируйте по убыванию цены

4. Посчитайте количество товаров с ценой > 20 через `=СЧЁТЕСЛИ(B2:B5;">20")`

6. Текстовый редактор: поиск и замена

Задание №10: текстовый поиск

В ЕГЭ дают текстовый файл. Нужно найти количество вхождений слова или сочетания.

Инструменты:

- **Блокнот:** Ctrl+F → «Найти далее» / «Подсчитать»
- **Word:** Ctrl+F → «Выделение чтения» → «Выделить всё»
- **VS Code:** Ctrl+F → видно количество совпадений

Регулярные выражения (базово)

Иногда нужно найти не точное слово, а шаблон. Для этого в продвинутых редакторах (VS Code, Notepad++) есть регулярные выражения.

Шаблон	Значение
`\d`	Любая цифра
`\w`	Любая буква или цифра
`.`	Любой символ
`*`	0 или более повторений
`+`	1 или более повторений
`[А-Я]`	Любая заглавная русская буква

Пример: найти все слова, начинающиеся на «про»:

- Шаблон: `\bпро\w*`
- Включить галочку «Регулярные выражения» при поиске

Домашнее задание

1. Python

- Установить Python и IDE (если ещё нет)
- Написать программу, которая запрашивает 3 числа и выводит их среднее арифметическое
- Написать программу, которая переводит градусы Цельсия в Фаренгейты: `F = C * 9/5 + 32`

Пример решения:

```python
# Среднее арифметическое трёх чисел
a = float(input("Введите первое число: "))
b = float(input("Введите второе число: "))
c = float(input("Введите третье число: "))
avg = (a + b + c) / 3
print(f"Среднее арифметическое: {avg:.2f}")

# Конвертер температуры
celsius = float(input("Введите градусы Цельсия: "))
fahrenheit = celsius * 9/5 + 32
print(f"{celsius}°C = {fahrenheit}°F")
```

2. Excel

- Создать таблицу из 10 товаров с ценой и количеством
- Посчитать общую сумму через СУММ
- Найти самый дорогой товар через МАКС
- Посчитать количество товаров дешевле 100 рублей через СЧЁТЕСЛИ

3. Текстовый редактор

- Открыть любой текст
- Найти все вхождения слова «информатика»
- Попробовать поиск с регулярными выражениями (если редактор поддерживает)

На следующем занятии (Занятие 2)

Тема: Переменные, типы, ввод/вывод (углубление)

- Подробно разберём типы данных
- Научимся работать с f-строками
- Освоим преобразование типов для решения задач ЕГЭ
- Начнём решать задачи №2 и №5

Карта курса (для справки)

```
Занятие 1:   Введение и инструменты ← мы здесь
Занятия 2-7: База Python
Занятия 8-13: Брутфорс
Занятия 14-15: Системы счисления и строки
Занятия 16-20: Рекурсия
Занятия 21-23: Теория игр
Занятия 24-27: Динамическое программирование
Занятия 28-29: Файлы и данные
Занятия 30-32: Excel/Calc и ручные задачи
Занятие 33:   Стратегия и пробник
```

Чек-лист первого занятия

- ☐

Установлен Python 3.9+

- ☐ Установлена IDE (PyCharm / VS Code)
- ☐ Установлен Jupyter Notebook
- ☐ Написана и запущена первая программа
- ☐ Освоены базовые операции в Excel
- ☐ Понятен принцип текстового поиска
- ☐ Выполнено домашнее задание

Глава 1. База Python

Занятие 2. Переменные, типы, ввод/вывод

План занятия

1. Типы данных: `int`, `float`, `str`, `bool`
2. Переменные и присваивание
3. Арифметические операции
4. Преобразование типов
5. Ввод данных: `input()`
6. Вывод данных: `print()` и f-строки
7. Практика

> **Цель занятия:** Научиться работать с данными разных типов, понимать преобразования и оформлять ввод/вывод. Это фундамент для ВСЕХ задач ЕГЭ.

1. Типы данных в Python

`int` — целые числа

Целые числа без ограничения по размеру (в отличие от многих других языков).

```
a = 42
b = -17
c = 0
big = 10**100  # Огромное число — Python справится

print(a, b, c)
print(f"10 в степени 100 = {big}")
```

`float` — дробные числа

Числа с плавающей точкой. Важно помнить про погрешность.

```
pi = 3.14159
e = 2.71828
half = 0.5
sci = 1.5e6  # 1.5 × 10⁶ = 1500000.0

print(f"pi = {pi}")
print(f"1.5e6 = {sci}")
```

> **Ловушка:** `0.1 + 0.2` не равно `0.3` из-за погрешности. На ЕГЭ это редко мешает, но помните.

```
print(0.1 + 0.2)       # 0.30000000000000004
print(0.1 + 0.2 == 0.3) # False!
```

`str` — строки

Последовательность символов. Всё, что в кавычках.

```
s1 = "Hello"
s2 = 'Мир'
s3 = "123"    # Это строка, не число!
s4 = ""       # Пустая строка

print(s1, s2)
print(f"s3 — это {type(s3)}")
```

`bool` — логический тип

Только два значения: `True` (истина) и `False` (ложь).

```
flag1 = True
flag2 = False

print(f"True — это {int(True)}")   # 1
print(f"False — это {int(False)}") # 0
```

> **Важно для ЕГЭ:** В логических задачах (№2) `True` и `False` можно использовать как `1` и `0`.

2. Переменные и присваивание

Что такое переменная

Переменная — это **именованная ячейка памяти**, в которой хранится значение.

```
x = 10        # Создали переменную x, положили 10
x = 20        # Перезаписали — теперь там 20
x = x + 5     # Взяли старое значение, прибавили 5, записали обратно

print(x)      # 25
```

Правила именования

- Буквы, цифры, знак подчёркивания `_`
- Не может начинаться с цифры
- Регистр важен: `name` и `Name` — разные переменные
- Не использовать ключевые слова: `if`, `for`, `while`, `def` и др.

```
# Хорошие имена
count = 0
max_value = 100
user_name = "Иван"
_temp = 42
```

```
# Плохие имена
# 1var = 10     # Ошибка! Начинается с цифры
# if = 5        # Ошибка! Ключевое слово
# my-var = 10   # Ошибка! Дефис недопустим
```

Множественное присваивание

```
a, b = 10, 20
print(f"a = {a}, b = {b}")

# Обмен значений без временной переменной
a, b = b, a
print(f"После обмена: a = {a}, b = {b}")
```

3. Арифметические операции

Полная таблица операций

Оператор	Операция	Пример	Результат
`+`	Сложение	`5 + 3`	`8`
`-`	Вычитание	`5 - 3`	`2`
`*`	Умножение	`5 * 3`	`15`
`/`	Деление	`5 / 2`	`2.5`
`//`	Целочисленное деление	`5 // 2`	`2`
`%`	Остаток от деления	`5 % 2`	`1`
`**`	Возведение в степень	`5 ** 3`	`125`

```
a, b = 17, 5

print(f"{a} + {b} = {a + b}")
print(f"{a} - {b} = {a - b}")
print(f"{a} * {b} = {a * b}")
print(f"{a} / {b} = {a / b}")      # Всегда float
print(f"{a} // {b} = {a // b}")    # Целая часть
print(f"{a} % {b} = {a % b}")      # Остаток
print(f"{a} ** {b} = {a ** b}")    # Степень
```

Остаток и целочисленное деление на ЕГЭ

Эти операции постоянно нужны:

```
# Проверка чётности (задания №5, 17, 25)
n = 42
if n % 2 == 0:
    print("Чётное")

# Последняя цифра числа (задание №25)
n = 12345
last_digit = n % 10       # 5
```

```
# Отбрасывание последней цифры
rest = n // 10              # 1234

# Проверка кратности (задание №17, 25, 27)
if n % 7 == 0:
    print("Кратно 7")

print(f"Последняя цифра: {last_digit}")
print(f"Число без последней цифры: {rest}")
```

Приоритет операций

1. `**` — степень
2. `*`, `/`, `//`, `%` — умножение, деление
3. `+`, `-` — сложение, вычитание

```
print(2 + 3 * 4)      # 14, не 20
print((2 + 3) * 4)    # 20 — скобки меняют порядок
print(2 ** 3 ** 2)    # 512 — вычисляется справа налево: 2 ** (3 ** 2) = 2
** 9
```

4. Преобразование типов

Основные функции преобразования

Функция	Назначение	Пример	Результат
`int(x)`	В целое число	`int("42")`	`42`
`int(x)`	В целое число	`int(3.9)`	`3` (отбрасывает дробную часть!)
`float(x)`	В дробное число	`float("3.14")`	`3.14`
`float(x)`	В дробное число	`float(5)`	`5.0`
`str(x)`	В строку	`str(42)`	`"42"`
`bool(x)`	В логическое	`bool(0)`	`False`
`bool(x)`	В логическое	`bool(42)`	`True`

```
# Преобразование строки в число
s = "123"
n = int(s)
print(f"Строка '{s}' → число {n}, тип: {type(n)}")

# Преобразование числа в строку
num = 456
text = str(num)
print(f"Число {num} → строка '{text}', тип: {type(text)}")

# int из float — ОТБРАСЫВАЕТ дробную часть, не округляет!
print(f"int(3.9) = {int(3.9)}")   # 3
print(f"int(-3.9) = {int(-3.9)}") # -3
```

Возможные ошибки

```
# Нельзя преобразовать не-числовую строку в число
# int("abc")  # ValueError!

# Нельзя преобразовать строку с точкой через int
# int("3.14")  # ValueError! Сначала float, потом int
```

Правильный путь: строка с точкой → число

```
s = "3.14"
n = float(s)       # 3.14 — дробное
m = int(float(s))  # 3 — целое (если нужно)

print(f"float('{s}') = {n}")
print(f"int(float('{s}')) = {m}")
```

5. Ввод данных: `input()`

Функция `input()`

`input()` **всегда возвращает строку**, даже если пользователь ввёл число.

```
# Базовый ввод
name = input("Введите ваше имя: ")
print(f"Привет, {name}!")

# Ввод числа — нужно преобразование!
age_str = input("Введите возраст: ")  # age_str — строка
age = int(age_str)                     # age — число
print(f"Через 5 лет будет {age + 5}")

# Совмещённая запись (часто используется на ЕГЭ)
n = int(input("Введите число: "))
print(f"Квадрат числа: {n ** 2}")
```

Ввод нескольких чисел

```
# Ввод двух чисел по одному
a = int(input("a = "))
b = int(input("b = "))
print(f"{a} + {b} = {a + b}")

# Ввод нескольких чисел в одной строке
data = input("Введите три числа через пробел: ").split()
a, b, c = int(data[0]), int(data[1]), int(data[2])
print(f"Сумма: {a + b + c}")

# Компактная запись через map (будет полезна в №17, 27)
a, b, c = map(int, input("Введите три числа через пробел: ").split())
print(f"Сумма: {a + b + c}")
```

6. Вывод данных: `print()` и f-строки

Функция `print()`

```
# Простой вывод
print("Hello")

# Вывод нескольких значений через пробел (по умолчанию)
print("Ответ:", 42, "балла")

# Разделитель (sep)
print("a", "b", "c", sep=", ")   # a, b, c

# Конец строки (end) — по умолчанию перевод строки \n
print("Первая строка", end=" → ")
print("Вторая строка")              # Первая строка → Вторая строка
```

f-строки — основной инструмент на ЕГЭ

f-строки позволяют вставлять значения переменных прямо в строку.

```
name = "Анна"
score = 88

# Простая вставка
print(f"Ученица {name} набрала {score} баллов")

# Выражения внутри f-строк
print(f"Сумма: {10 + 20}")
print(f"Квадрат 5: {5 ** 2}")
```

Форматирование в f-строках

```
pythonpi

# Округление до 2 знаков
print(f"π ≈ {pi:.2f}")

# Выравнивание по ширине
for i in range(1, 4):
    print(f"Число: {i:5d} Квадрат: {i**2:5d}")

# Двоичное представление (пригодится в №5, 14)
n = 42
print(f"{n} в двоичной: {n:b}")
print(f"{n} в восьмеричной: {n:o}")
print(f"{n} в шестнадцатеричной: {n:x}")
```

> **Совет:** На ЕГЭ всегда используйте f-строки. Они читаемее и быстрее, чем `.format()` или `%`.

7. Практика

Задача 1. Сумма и произведение

Напишите программу, которая запрашивает два целых числа и выводит их сумму и произведение.

```
a = int(input("Введите первое число: "))
b = int(input("Введите второе число: "))

print(f"Сумма: {a + b}")
print(f"Произведение: {a * b}")
```

Задача 2. Последняя цифра

Дано натуральное число. Выведите его последнюю цифру.

```
n = int(input("Введите число: "))
last = n % 10
print(f"Последняя цифра: {last}")
```

Задача 3. Перевод градусов

Переведите температуру из градусов Цельсия в градусы Фаренгейта по формуле: `F = C × 9/5 + 32`.

```
c = float(input("Введите градусы Цельсия: "))
f = c * 9 / 5 + 32
print(f"{c}°C = {f:.1f}°F")
```

Задача 4. Сумма цифр трёхзначного числа

Дано трёхзначное число. Найдите сумму его цифр.

```
n = int(input("Введите трёхзначное число: "))

d1 = n // 100        # Первая цифра
d2 = (n // 10) % 10 # Вторая цифра
d3 = n % 10          # Третья цифра

print(f"Сумма цифр: {d1 + d2 + d3}")
```

Задача 5. Проверка кратности (стиль задания №17)

Дано число. Проверьте, кратно ли оно 7 и не кратно ли 5.

```
n = int(input("Введите число: "))

if n % 7 == 0 and n % 5 ≠ 0:
    print("Число кратно 7 и не кратно 5")
else:
    print("Условие не выполнено")
```

Домашнее задание

Задание 1. Калькулятор

Напишите программу, которая запрашивает два числа и операцию (`+` , `-` , `*` , `/`) и выводит результат.

```
# Шаблон:
a = float(input("Первое число: "))
op = input("Операция (+, -, *, /): ")
b = float(input("Второе число: "))

# Ваш код здесь
```

Задание 2. Обратное число

Дано трёхзначное число. Выведите число, полученное при прочтении его цифр справа налево.

```
# Пример:
# Ввод: 123
# Вывод: 321

n = int(input())
# Ваш код здесь
```

Задание 3. Площадь круга

Запросите радиус круга и выведите его площадь с точностью до 3 знаков после запятой. Используйте `pi` из модуля `math` .

```
import math

r = float(input("Радиус: "))
# Ваш код здесь (S = π * r²)
```

Задание 4. Чётность и кратность

Дано число. Выведите:

- «Чётное» или «Нечётное»
- «Кратно 3» или «Не кратно 3»

```
n = int(input())
# Ваш код здесь
```

Что мы сегодня изучили

Тема	Ключевые моменты
Типы данных	`int`, `float`, `str`, `bool`
Переменные	Именование, присваивание, обмен
Арифметика	`+`, `-`, `*`, `/`, `//`, `%`, `**`
Преобразование	`int()`, `float()`, `str()`
Ввод	`input()` всегда возвращает строку
Вывод	`print()`, f-строки, форматирование

> **Связь с ЕГЭ:** Эти знания используются **во всех задачах без исключения**. Особенно важны: остаток `%`, целочисленное деление `//`, преобразование типов, f-строки.

На следующем занятии (Занятие 3)

Тема: Условия и ветвления

- `if`, `elif`, `else`
- Логические операторы: `and`, `or`, `not`
- Вложенные условия
- Решение задач №2 и №6

Чек-лист занятия

- ☐ Понимаю разницу между `int`, `float`, `str`, `bool`
- ☐ Умею преобразовывать типы (`int()`, `str()`, `float()`)
- ☐ Знаю, что `input()` всегда возвращает строку
- ☐ Умею использовать `//` и `%` для выделения цифр
- ☐ Владею f-строками для форматирования вывода
- ☐ Выполнено домашнее задание

Занятие 3. Условия и ветвления

План занятия

1. Операторы сравнения

2. Логические операторы: `and`, `or`, `not`
3. Условный оператор: `if`, `elif`, `else`
4. Вложенные условия
5. Приоритет операций
6. Практика

> **Цель занятия:** Научиться управлять потоком программы. Условия — основа для задач №2, 5, 6, 12, 15, 17, 25.

1. Операторы сравнения

Таблица операторов сравнения

Результат сравнения — **всегда** **`bool`** (`True` или `False`).

Оператор	Значение	Пример	Результат
`==`	Равно	`5 == 5`	`True`
`!=`	Не равно	`5 != 3`	`True`
`<`	Меньше	`3 < 5`	`True`
`>`	Больше	`5 > 3`	`True`
`<=`	Меньше или равно	`5 <= 5`	`True`
`>=`	Больше или равно	`5 >= 3`	`True`

```
a, b = 10, 20

print(f"{a} == {b}: {a == b}")   # False
print(f"{a} != {b}: {a != b}")   # True
print(f"{a} < {b}: {a < b}")     # True
print(f"{a} > {b}: {a > b}")     # False
print(f"{a} <= {b}: {a <= b}")   # True
print(f"{a} >= {b}: {a >= b}")   # False
```

Сравнение дробных чисел

Из-за погрешности `float` прямое сравнение может подвести.

```
x = 0.1 + 0.2
y = 0.3

print(f"x = {x}")
print(f"y = {y}")
print(f"x == y: {x == y}")      # False!

# Правильный способ — сравнение с погрешностью
eps = 1e-9
print(f"|x - y| < eps: {abs(x - y) < eps}")  # True
```

> **Для ЕГЭ:** В задачах экзамена обычно используются целые числа, поэтому проблема `float` встречается редко. Но помните о ней.

2. Логические операторы: `and`, `or`, `not`

Таблица логических операторов

Оператор	Значение	Пример	Результат
`and`	И (оба истинны)	`True and False`	`False`
`or`	ИЛИ (хотя бы одно истинно)	`True or False`	`True`
`not`	НЕ (отрицание)	`not True`	`False`

Таблицы истинности

`and` (конъюнкция):

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	**True**

`or` (дизъюнкция):

A	B	A or B
False	False	False
False	True	**True**
True	False	**True**
True	True	**True**

`not` (отрицание):

A	not A
False	True
True	False

```
# Примеры логических операций
x, y = 5, 10

# and — оба условия должны быть истинны
print(f"x > 0 and y > 0: {x > 0 and y > 0}")      # True
print(f"x > 0 and y < 0: {x > 0 and y < 0}")      # False

# or — хотя бы одно условие истинно
```

```
print(f"x > 0 or y < 0: {x > 0 or y < 0}")       # True
print(f"x < 0 or y < 0: {x < 0 or y < 0}")       # False

# not — отрицание
print(f"not (x > 0): {not (x > 0)}")             # False
print(f"not (x < 0): {not (x < 0)}")             # True
```

Типовые условия на ЕГЭ

```
# Проверка диапазона (задание №17, 25)
x = 15
if 10 ⩽ x ⩽ 20:   # Эквивалентно: x ⩾ 10 and x ⩽ 20
    print("x в диапазоне [10, 20]")

# Проверка кратности (задание №17, 25, 27)
n = 42
if n % 2 == 0 and n % 3 == 0:
    print("Кратно 2 и 3 (кратно 6)")

# Проверка НЕравенства
if n % 5 ≠ 0:
    print("НЕ кратно 5")

# Исключающее ИЛИ через ≠ (задание №2)
a, b = True, False
if a ≠ b:  # Истинно, когда значения разные
    print("Ровно одно истинно (исключающее ИЛИ)")
```

Короткое замыкание (short-circuit)

Python вычисляет логические выражения слева направо и останавливается, как только результат определён.

```
# and: если первый операнд False, второй не вычисляется
False and print("Это не выведется")

# or: если первый операнд True, второй не вычисляется
True or print("И это не выведется")

# Это можно использовать для защиты от ошибок
s = None
if s is not None and len(s) > 0:  # Второе условие не проверится, если s
is None
    print("Строка не пуста")
```

3. Условный оператор: `if`, `elif`, `else`

Базовая конструкция

```
if условие:
    блок кода (выполняется, если условие истинно)
elif другое_условие:
    блок кода (выполняется, если первое ложно, а это истинно)
```

```
    else:
        блок кода (выполняется, если всё ложное)
```

```python
age = 20

if age < 18:
    print("Несовершеннолетний")
elif age < 65:
    print("Взрослый")
else:
    print("Пенсионер")
```

Синтаксис

- **Двоеточие** `:` после условия обязательно
- **Отступы** — 4 пробела (или 1 таб) для блока кода
- `elif` может быть сколько угодно (или ни одного)
- `else` может быть только один (или ни одного)

```python
# Простое if (без elif и else)
x = 42
if x > 0:
    print("Положительное")

# if-else (два варианта)
if x % 2 == 0:
    print("Чётное")
else:
    print("Нечётное")

# if-elif-else (много вариантов)
score = 85
if score >= 90:
    grade = "Отлично"
elif score >= 75:
    grade = "Хорошо"
elif score >= 60:
    grade = "Удовлетворительно"
else:
    grade = "Неудовлетворительно"
print(f"Оценка: {grade}")
```

Тернарный оператор (компактное if-else)

Полезно, когда нужно присвоить значение в зависимости от условия.

```python
# Обычный if-else
x = 10
if x > 0:
    sign = "+"
else:
    sign = "-"

# Тернарный оператор (в одну строку)
sign = "+" if x > 0 else "-"
```

```
print(f"Знак: {sign}")

# Можно вкладывать (но осторожно — читаемость страдает)
x = 0
sign = "+" if x > 0 else ("-" if x < 0 else "0")
print(f"Знак: {sign}")
```

4. Вложенные условия

Условия внутри условий

```
x, y = 5, 3

if x > 0:
    if y > 0:
        print("Первая четверть")
    else:
        print("Четвёртая четверть")
else:
    if y > 0:
        print("Вторая четверть")
    else:
        print("Третья четверть")
```

Вложенные условия vs логические операторы

Часто вложенные условия можно заменить на `and` / `or`.

```
# Вложенные условия (менее читаемо)
if x > 0:
    if y > 0:
        print("x и y положительные")

# Логический оператор (более читаемо)
if x > 0 and y > 0:
    print("x и y положительные")
```

> **Совет:** Если вложенность больше 2 уровней, задумайтесь — возможно, код можно упростить.

5. Приоритет операций (полная картина)

Приоритет от высшего к низшему

Приоритет	Операторы	Описание
1 (высший)	`()`	Скобки
2	`**`	Возведение в степень
3	`+x`, `-x`, `~x`	Унарные плюс, минус, побитовое НЕ
4	`*`, `/`, `//`, `%`	Умножение, деление
5	`+`, `-`	Сложение, вычитание
6	`<<`, `>>`	Побитовый сдвиг
7	`&`	Побитовое И
8	`^`	Побитовое исключающее ИЛИ
9	`|`	Побитовое ИЛИ
10	`==`, `!=`, `<`, `>`, `<=`, `>=`, `is`, `in`	Сравнения
11	`not`	Логическое НЕ
12	`and`	Логическое И
13 (низший)	`or`	Логическое ИЛИ

Ключевые правила

1. **`not` связывает сильнее, чем `and`**
2. **`and` связывает сильнее, чем `or`**
3. **Скобки всегда надёжнее** — если сомневаетесь, ставьте скобки

```
# Приоритет not > and > or
print(not False and False)   # (not False) and False = True and False = False
print(not (False and False)) # not (False) = True

print(True or True and False)
# True or (True and False) = True or False = True
print((True or True) and False) # True and False = False

# Сравнение vs логика: сравнение приоритетнее
print(5 > 3 and 2 < 4)  # (5 > 3) and (2 < 4) = True and True = True
```

Типичная ошибка

```
# НЕВЕРНО: 0 < x < 10 and y > 5 — правильно, Python это понимает
# НО: x > 0 and < 10 — ОШИБКА синтаксиса!

x = 5
# Правильно: двойное сравнение (Python поддерживает)
if 0 < x < 10:
```

```
    print("x в диапазоне (0, 10)")

# Эквивалентно:
if x > 0 and x < 10:
    print("x в диапазоне (0, 10)")
```

6. Практика

Задача 1. Проверка чётности

Дано целое число. Выведите «Чётное», если оно чётное, и «Нечётное» в противном случае.

```
n = int(input("Введите число: "))

if n % 2 == 0:
    print("Чётное")
else:
    print("Нечётное")
```

Задача 2. Определение четверти координатной плоскости

Даны координаты точки (x, y), не лежащей на осях. Определите номер четверти.

```
x = float(input("Введите x: "))
y = float(input("Введите y: "))

if x > 0 and y > 0:
    print("Первая четверть")
elif x < 0 and y > 0:
    print("Вторая четверть")
elif x < 0 and y < 0:
    print("Третья четверть")
elif x > 0 and y < 0:
    print("Четвёртая четверть")
else:
    print("Точка на оси")
```

Задача 3. Решение квадратного уравнения

Даны коэффициенты a, b, c квадратного уравнения $ax^2 + bx + c = 0$. Найдите его корни.

```
import math

a = float(input("Введите a: "))
b = float(input("Введите b: "))
c = float(input("Введите c: "))

if a == 0:
    # Линейное уравнение bx + c = 0
    if b == 0:
        if c == 0:
            print("Бесконечно много решений")
        else:
            print("Решений нет")
```

```
    else:
        x = -c / b
        print(f"Один корень: x = {x:.3f}")
else:
    D = b**2 - 4*a*c
    if D > 0:
        x1 = (-b + math.sqrt(D)) / (2*a)
        x2 = (-b - math.sqrt(D)) / (2*a)
        print(f"Два корня: x₁ = {x1:.3f}, x₂ = {x2:.3f}")
    elif D == 0:
        x = -b / (2*a)
        print(f"Один корень: x = {x:.3f}")
    else:
        print("Корней нет (дискриминант < 0)")
```

Задача 4. Классификация треугольника

Даны три стороны треугольника. Определите, является ли он равносторонним, равнобедренным или разносторонним.

```
a = float(input("Сторона a: "))
b = float(input("Сторона b: "))
c = float(input("Сторона c: "))

# Проверка существования треугольника
if a + b > c and b + c > a and a + c > b:
    if a == b == c:
        print("Равносторонний")
    elif a == b or b == c or a == c:
        print("Равнобедренный")
    else:
        print("Разносторонний")
else:
    print("Треугольник не существует")
```

Задача 5. Високосный год (стиль задания №6)

Дано целое число — год. Определите, является ли он високосным.

Правило: Год високосный, если:

- делится на 400 **ИЛИ**
- делится на 4 **И** не делится на 100

```
year = int(input("Введите год: "))

if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    print("Високосный")
else:
    print("Не високосный")
```

Задача 6. Попадание в отрезок (стиль задания №15)

Даны число x и границы отрезка [A, B]. Проверьте, принадлежит ли x отрезку.

```python
x = float(input("x = "))
A = float(input("A = "))
B = float(input("B = "))

# Проверка с учётом, что отрезок может быть [A, B] или [B, A]
left = min(A, B)
right = max(A, B)

if left ⩽ x ⩽ right:
    print(f"{x} ∈ [{left}, {right}]")
else:
    print(f"{x} ∉ [{left}, {right}]")
```

Домашнее задание

Задание 1. Максимум из трёх

Даны три целых числа. Найдите максимальное (не используя `max()`).

```python
a = int(input("a = "))
b = int(input("b = "))
c = int(input("c = "))

# Ваш код здесь
```

Задание 2. Описание возраста

Дано число — возраст. Выведите:

- «Ребёнок» (0–12)
- «Подросток» (13–17)
- «Взрослый» (18–64)
- «Пенсионер» (65+)

```python
age = int(input("Возраст: "))

# Ваш код здесь
```

Задание 3. Ближайший к 10

Даны два числа. Выведите то, которое ближе к 10. Если одинаково — выведите любое.

```python
a = float(input("a = "))
b = float(input("b = "))

# Ваш код здесь (используйте abs())
```

Задание 4. Точка в прямоугольнике

Дан прямоугольник с углами (x1, y1) и (x2, y2) и точка (x, y). Проверьте, лежит ли точка внутри прямоугольника (включая границы).

```
x1, y1 = map(float, input("Левый нижний угол (x1 y1): ").split())
x2, y2 = map(float, input("Правый верхний угол (x2 y2): ").split())
x, y = map(float, input("Точка (x y): ").split())

# Ваш код здесь
```

Что мы сегодня изучили

Тема	Ключевые моменты
Операторы сравнения	`==` , `!=` , `<` , `>` , `<=` , `>=`
Логические операторы	`and` , `or` , `not`
`if-elif-else`	Двоеточие, отступы, порядок проверки
Вложенные условия	До 2 уровней, лучше заменять на `and`
Приоритет операций	`not` > `and` > `or` , скобки — лучший друг

> **Связь с ЕГЭ:** Условия используются буквально в каждой задаче. Особенно важны: проверка кратности (`%`), проверка диапазона, составные условия с `and` / `or` .

На следующем занятии (Занятие 4)

Тема: Циклы

- `for i in range()`
- `while`
- `break` , `continue`
- Вложенные циклы
- Решение задач №5, 6, 12

Чек-лист занятия

- ☐ Знаю все операторы сравнения
- ☐ Понимаю таблицы истинности `and` , `or` , `not`
- ☐ Умею писать `if-elif-else` с правильными отступами
- ☐ Понимаю приоритет `not` > `and` > `or`
- ☐ Могу заменить вложенные условия на `and`
- ☐ Выполнено домашнее задание

Занятие 4. Циклы

План занятия

1. Цикл `for i in range()`
2. Цикл `while`
3. `break` и `continue`
4. Вложенные циклы
5. Бесконечный цикл и как его избежать
6. Практика

> **Цель занятия:** Освоить главный инструмент перебора. Циклы — основа для задач №5, 6, 8, 12, 14, 17, 24, 25, 27.

1. Цикл `for i in range()`

Функция `range()`

`range()` создаёт последовательность чисел. **Не список** — это генератор, который выдаёт числа по одному.

```
range(stop)              → 0, 1, 2, ..., stop-1
range(start, stop)       → start, start+1, ..., stop-1
range(start, stop, step) → start, start+step, ..., (не доходя до stop)
```

```
# range(stop) — от 0 до stop-1
print("range(5):", list(range(5)))      # [0, 1, 2, 3, 4]

# range(start, stop) — от start до stop-1
print("range(2, 7):", list(range(2, 7)))  # [2, 3, 4, 5, 6]

# range(start, stop, step) — с шагом
print("range(1, 10, 2):", list(range(1, 10, 2)))  # [1, 3, 5, 7, 9]

# Отрицательный шаг — обратный порядок
print("range(10, 0, -1):", list(range(10, 0, -1)))  # [10, 9, ..., 1]
```

Цикл `for`

```
# Простой перебор
for i in range(5):
    print(i, end=" ")  # 0 1 2 3 4
print()

# Перебор с шагом
for i in range(0, 10, 2):
    print(i, end=" ")  # 0 2 4 6 8
print()
```

```python
# Перебор в обратном порядке
for i in range(5, 0, -1):
    print(i, end=" ")  # 5 4 3 2 1
print()
```

Типовые паттерны `for` на ЕГЭ

```python
# Суммирование (задания №17, 27)
total = 0
for i in range(1, 101):
    total += i
print(f"Сумма 1..100 = {total}")

# Подсчёт с условием (задания №17, 25)
count = 0
for i in range(1, 101):
    if i % 7 == 0:
        count += 1
print(f"Чисел, кратных 7, от 1 до 100: {count}")

# Поиск максимума (задания №17, 24, 27)
max_val = -10**9  # Очень маленькое начальное значение
for i in [3, 7, 1, 9, 4, 6, 8]:
    if i > max_val:
        max_val = i
print(f"Максимум: {max_val}")
```

Важно: `range(start, stop)` не включает `stop`

Это частая ошибка новичков.

```python
# Хотим числа от 1 до 10 включительно
for i in range(1, 11):  # stop = 11, а не 10!
    print(i, end=" ")
print()

# Или так:
for i in range(1, 10 + 1):  # Явно показываем, что включаем 10
    print(i, end=" ")
print()
```

Перебор без `range()` — по элементам коллекции

```python
# Перебор символов строки
for char in "Python":
    print(char, end=" ")
print()

# Перебор элементов списка
for num in [10, 20, 30, 40]:
    print(num, end=" ")
print()

# Перебор с индексом: enumerate()
```

```python
for i, char in enumerate("ABC"):
    print(f"Индекс {i}: {char}")
```

2. Цикл `while`

Синтаксис

```
while условие:
    блок кода (выполняется, пока условие истинно)
```

`while` нужен, когда **заранее неизвестно**, сколько будет итераций.

```python
# Суммируем числа, пока не наберём 100
s = 0
i = 1
while s < 100:
    s += i
    i += 1
print(f"Понадобилось {i-1} чисел, сумма = {s}")
```

`for` vs `while` — когда что использовать

Ситуация	Что выбрать
Известно количество итераций	`for i in range(N)`
Неизвестно количество итераций	`while условие:`
Перебор элементов списка/строки	`for item in collection`
Поиск с неизвестным концом	`while` + `break`

`while` на ЕГЭ

```python
# Задание №12: редактор строк — while с заменой
s = "ПОПОЛАМ"
while "ПО" in s:
    s = s.replace("ПО", "АП", 1)
print(s)

# Задание №24: поиск максимальной подстроки
s = "ABCABABC"
max_len = 0
cur_len = 0
for char in s:
    if char ≠ "A":
        cur_len += 1
        max_len = max(max_len, cur_len)
    else:
        cur_len = 0
print(f"Максимальная длина подстроки без 'A': {max_len}")

# Задание №5: моделирование алгоритма
n = 10
while n > 1:
```

```
    if n % 2 == 0:
        n = n // 2
    else:
        n = 3 * n + 1
    print(n, end=" ")
print()
```

Опасность: бесконечный цикл

```
# ОПАСНО: условие всегда истинно — бесконечный цикл!
# while True:
#     print("Это никогда не кончится")

# Правильно: условие должно когда-то стать ложным
i = 0
while i < 5:
    print(i, end=" ")
    i += 1  # Без этой строки — бесконечный цикл
print()
```

3. `break` и `continue`

`break` — немедленный выход из цикла

```
# Поиск первого отрицательного числа
numbers = [3, 7, -1, 9, -5, 4]

for num in numbers:
    if num < 0:
        print(f"Первое отрицательное: {num}")
        break  # Выходим из цикла
    print(f"Проверили {num}")
```

`continue` — пропустить остаток итерации, перейти к следующей

```
# Сумма положительных чисел
total = 0
for num in [3, -2, 7, -5, 1]:
    if num < 0:
        continue  # Пропускаем отрицательные
    total += num
    print(f"Добавили {num}, сумма = {total}")
print(f"Итоговая сумма: {total}")
```

`break` и `continue` на ЕГЭ

```
# Задание №25: поиск чисел, прерывание при превышении
for num in range(100, 200):
    if num % 17 == 0:
        print(f"Первое кратное 17: {num}")
        break  # Дальше не ищем

# Пропуск неинтересных значений
```

```python
for i in range(100):
    if i % 2 == 0:
        continue  # Пропускаем чётные
    # Обрабатываем только нечётные
```

`break` выходит только из ОДНОГО цикла

```python
for i in range(3):
    for j in range(3):
        if i == j == 1:
            print(f"break на i={i}, j={j}")
            break  # Выходит только из внутреннего цикла по j
    print(f"Внешний цикл: i={i}")
# Внешний цикл продолжает работать!
```

4. Вложенные циклы

Цикл внутри цикла

Внутренний цикл полностью выполняется на каждой итерации внешнего.

```python
# Таблица умножения 3×3
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} × {j} = {i*j}", end="  ")
    print()  # Перевод строки после каждой строки таблицы
```

Вложенные циклы на ЕГЭ

```python
# Задание №2: перебор всех наборов переменных
for x in [0, 1]:
    for y in [0, 1]:
        for z in [0, 1]:
            F = (x or y) and not z
            print(f"{x} {y} {z} → {int(F)}")

# Задание №8: перебор слов (упрощённо)
count = 0
for a in "ABC":
    for b in "ABC":
        for c in "ABC":
            word = a + b + c
            if "AA" not in word:
                count += 1
print(f"Слов без 'AA': {count}")

# Задание №14: перебор x для разных оснований
for base in range(3, 10):
    for x in range(base):
        # Проверка условия для данного x и base
        pass
```

Оценка сложности вложенных циклов

```
# Один цикл: N итераций
# Два вложенных: N² итераций — при N=1000 это уже 1 000 000
# Три вложенных: N³ итераций — при N=100 это 1 000 000

# На ЕГЭ избегайте трёх вложенных циклов с большим диапазоном!
# Вместо этого используйте itertools.product()
```

5. Бесконечный цикл и как его избежать

`while True`

Это намеренный бесконечный цикл. Из него выходят только через `break`.

```
# Поиск с неизвестным количеством шагов
n = 100
steps = 0
while True:
    if n == 1:
        break
    if n % 2 == 0:
        n = n // 2
    else:
        n = 3 * n + 1
    steps += 1
print(f"Достигли 1 за {steps} шагов")
```

Три причины нежелательного бесконечного цикла

1. Забыли изменить переменную условия:

```
# ОШИБКА: i никогда не меняется
# i = 0
# while i < 5:
#     print(i)
#     # i += 1  # Забыли!
```

2. Условие никогда не станет ложным:

```
# ОШИБКА: условие всегда True
# x = 10
# while x > 0:
#     x = x + 1  # x только растёт!
```

3. Ошибка в логике условия:

```
# ОШИБКА: перепутали and/or
# while x > 0 and x < 100:  # OK
# while x > 0 or x < 100:   # Всегда True!
```

Защита от бесконечного цикла

```
# Всегда добавляйте ограничитель итераций
max_iterations = 10000
iter_count = 0

while условие:
    # ... код ...
    iter_count += 1
    if iter_count > max_iterations:
        print("Превышено максимальное число итераций!")
        break
```

6. Практика

Задача 1. Сумма чисел от 1 до N

Дано натуральное число N. Найдите сумму всех чисел от 1 до N.

```
n = int(input("Введите N: "))
total = 0
for i in range(1, n + 1):
    total += i
print(f"Сумма от 1 до {n} = {total}")
```

Задача 2. Таблица умножения

Выведите таблицу умножения от 1 до 9 (как в школьной тетради).

```
for i in range(1, 10):
    for j in range(1, 10):
        print(f"{i} × {j} = {i*j:2d}", end="  ")
    print()
```

Задача 3. Поиск максимума в последовательности

С клавиатуры вводятся числа, ввод заканчивается нулём. Найдите максимальное.

```
max_val = -10**9

while True:
    x = int(input("Введите число (0 — конец): "))
    if x == 0:
        break
    if x > max_val:
        max_val = x

if max_val == -10**9:
    print("Не введено ни одного числа")
else:
    print(f"Максимум: {max_val}")
```

Задача 4. Количество делителей (стиль задания №25)

Дано число N. Подсчитайте количество его натуральных делителей.

```
n = int(input("Введите число: "))
count = 0

for i in range(1, n + 1):
    if n % i == 0:
        count += 1

print(f"Количество делителей: {count}")
```

> **Оптимизация:** Для больших чисел перебирают до $\sqrt{n}$. Но это тема занятия №13.

Задача 5. Сумма цифр числа (стиль задания №5)

Дано натуральное число. Найдите сумму его цифр.

```
n = int(input("Введите число: "))
total = 0

while n > 0:
    total += n % 10  # Добавляем последнюю цифру
    n = n // 10       # Отбрасываем последнюю цифру

print(f"Сумма цифр: {total}")
```

Задача 6. Проверка простоты числа

Дано число N > 1. Определите, простое ли оно.

```
n = int(input("Введите число: "))
is_prime = True

for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
        is_prime = False
        break

if is_prime:
    print("Простое")
else:
    print("Составное")
```

Домашнее задание

Задание 1. Факториал

Дано натуральное число N. Вычислите N! (произведение чисел от 1 до N).

```
n = int(input("Введите N: "))
# Ваш код здесь
```

Задание 2. Сумма чётных

Дано N. Найдите сумму всех чётных чисел от 1 до N.

```
n = int(input("Введите N: "))
# Ваш код здесь
```

Задание 3. Обратный порядок

Выведите числа от N до 1 в обратном порядке.

```
n = int(input("Введите N: "))
# Ваш код здесь (используйте range с шагом -1)
```

Задание 4. Подсчёт гласных

Дана строка. Подсчитайте количество гласных букв (а, е, ё, и, о, у, ы, э, ю, я).

```
s = input("Введите строку: ")
vowels = "аеёиоуыэюяАЕЁИОУЫЭЮЯ"
# Ваш код здесь
```

Задание 5. Числа Фибоначчи

Выведите первые N чисел Фибоначчи (1, 1, 2, 3, 5, 8, ...).

```
n = int(input("Введите N: "))
# Ваш код здесь (a, b = 1, 1)
```

Что мы сегодня изучили

Тема	Ключевые моменты
`for i in range()`	`range(start, stop, step)`, stop не включается
`while условие:`	Для неизвестного числа итераций
`break`	Немедленный выход из цикла
`continue`	Пропуск остатка итерации
Вложенные циклы	Внутренний выполняется полностью на каждой итерации внешнего
Бесконечный цикл	`while True` + `break`, всегда ограничитель итераций

> **Связь с ЕГЭ:** Циклы — основа почти всех задач. `for` используется для перебора (`№2, 5, 8, 14, 17, 25`), `while` — для редактора строк (`№12`) и неизвестного числа шагов (`№5, 6, 24`).

На следующем занятии (Занятие 5)

Тема: Строки

- Индексация, срезы `[::]`

- Методы: `.find()`, `.replace()`, `.split()`, `.join()`, `.count()`
- Решение задач №12, 24

Чек-лист занятия

- ☐ Знаю синтаксис `range(start, stop, step)`
- ☐ Понимаю разницу между `for` и `while`
- ☐ Умею использовать `break` и `continue`
- ☐ Могу написать вложенные циклы
- ☐ Знаю, как избежать бесконечного цикла
- ☐ Выполнено домашнее задание

Занятие 5. Строки

План занятия

1. Индексация строк
2. Срезы `[start:stop:step]`
3. Методы поиска: `.find()`, `.rfind()`, `.count()`
4. Методы проверки: `.startswith()`, `.endswith()`, `.isdigit()`, `.isalpha()`
5. Методы изменения: `.replace()`, `.upper()`, `.lower()`, `.split()`, `.join()`
6. Неизменяемость строк
7. Практика

> **Цель занятия:** Освоить работу со строками — ключевой навык для задач №5, 8, 12, 14, 24.

1. Индексация строк

Прямая индексация (слева направо)

Строка — это последовательность символов. Каждый символ имеет индекс, начиная с **0**.

```
 Строка:   П  р  и  в  е  т
 Индекс:   0  1  2  3  4  5
Длина = 6, индексы от 0 до 5
```

```
s = "Привет"

print(f"Строка: '{s}'")
```

```
print(f"Длина: {len(s)}")
print(f"s[0] = '{s[0]}'")   # П — первый символ
print(f"s[1] = '{s[1]}'")   # р
print(f"s[5] = '{s[5]}'")   # т — последний символ
print(f"s[-1] = '{s[-1]}'") # т — последний символ (отрицательный индекс)
```

Отрицательная индексация (справа налево)

```
 Строка:     П   р   и   в   е   т
 Индекс:     0   1   2   3   4   5
 Отрицат.:  -6  -5  -4  -3  -2  -1
```

```
s = "Привет"

print(f"s[-1] = '{s[-1]}'")  # т — последний
print(f"s[-2] = '{s[-2]}'")  # е — предпоследний
print(f"s[-6] = '{s[-6]}'")  # П — первый (len(s) = 6, -len(s) = -6)
```

Выход за границы — ошибка!

```
s = "Привет"
# print(s[6])   # IndexError! Последний индекс = 5
# print(s[-7])  # IndexError! Минимальный = -6
```

2. Срезы `[start:stop:step]`

Синтаксис среза

```
s[start:stop:step]
   start — с какого индекса начинаем (включительно)
   stop  — до какого индекса (НЕ включительно)
   step  — шаг (по умолчанию 1)
```

```
s = "Привет, мир!"

# s[start:stop] — от start до stop-1
print(f"s[0:6]   = '{s[0:6]}'")     # Привет
print(f"s[:6]    = '{s[:6]}'")      # От начала до 6: Привет (start=0 по
умолчанию)
print(f"s[7:]    = '{s[7:]}'")      # От 7 до конца: мир! (stop=len(s) по
умолчанию)
print(f"s[:]     = '{s[:]}'")       # Вся строка: Привет, мир!

# s[start:stop:step] — с шагом
print(f"s[::2]   = '{s[::2]}'")     # Каждый второй символ: П и е , м р
print(f"s[1:8:2] = '{s[1:8:2]}'")   # Индексы 1,3,5,7: р в т м

# Отрицательный шаг — обратный порядок
print(f"s[::-1]  = '{s[::-1]}'")    # Строка наоборот: !рим ,тевирП
```

Срезы на ЕГЭ

```
s = "информатика"

# Переворот строки (задание №12, 24)
```

```
reversed_s = s[::-1]
print(f"Перевёрнутая: {reversed_s}")

# Каждый второй символ (задание №5, 14)
every_second = s[::2]
print(f"Каждый второй: {every_second}")

# Удаление последнего символа
without_last = s[:-1]
print(f"Без последнего: {without_last}")

# Первые три символа
first_three = s[:3]
print(f"Первые три: {first_three}")

# Последние три символа
last_three = s[-3:]
print(f"Последние три: {last_three}")
```

Мнемоника: как запомнить срезы

```
s = "Python"

[ 0 : 3 ]  → "Pyt"   (индексы 0, 1, 2)
 ↑   ↑
start stop (не включая)

[ : : -1 ] → "nohtyP" (вся строка в обратном порядке)
     ↑
   шаг -1 = идём справа налево
```

3. Методы поиска

`.find()` — поиск подстроки

Возвращает **индекс первого вхождения** или **-1**, если не найдено.

```
s = "информатика"

print(f".find('о'):     {s.find('о')}")       # 2
print(f".find('форма'): {s.find('форма')}")   # 2
print(f".find('а'):     {s.find('а')}")       # 9 (первое 'а')
print(f".find('я'):     {s.find('я')}")       # -1 (нет такого символа)

# Поиск с указанной позиции
print(f".find('а', 5):  {s.find('а', 5)}")    # 9 (ищем с индекса 5)
```

`.rfind()` — поиск справа

Возвращает **индекс последнего вхождения**.

```
s = "информатика"
```

```
print(f".rfind('а'): {s.rfind('а')}")    # 9 (последнее 'а')
print(f".rfind('и'): {s.rfind('и')}")    # 8
```

`.count()` — подсчёт вхождений

```
s = "А роза упала на лапу Азора"

print(f".count('а'):   {s.count('а')}")     # 7 (все 'а', включая
заглавную? Нет!)
print(f".count('а'):   {s.lower().count('а')}")  # 8 (привели к нижнему
регистру)
print(f".count('ла'):  {s.count('ла')}")    # 2
print(f".count(' '):   {s.count(' ')}")     # 5 (пробелы)
```

Поиск на ЕГЭ

```
# Задание №12: проверка наличия подстроки
s = "ПОПОЛАМ"
while "ПО" in s:          # Быстрая проверка наличия
    s = s.replace("ПО", "АП", 1)
print(s)

# Задание №24: подсчёт символов
s = "АВСАВАВС"
print(f"Букв 'А': {s.count('А')}")

# Поиск позиции для среза
s = "начало-середина-конец"
pos = s.find('-')
first_part = s[:pos]  # "начало"
print(f"Первая часть: {first_part}")
```

4. Методы проверки

Обзор методов проверки

Метод	Возвращает `True`, если
`.startswith(s)`	Строка начинается с `s`
`.endswith(s)`	Строка заканчивается на `s`
`.isdigit()`	Все символы — цифры
`.isalpha()`	Все символы — буквы
`.isalnum()`	Все символы — буквы или цифры
`.isspace()`	Все символы — пробельные
`.islower()`	Все буквы строчные
`.isupper()`	Все буквы заглавные

```
print(f"'123'.isdigit():   {'123'.isdigit()}")     # True
print(f"'abc'.isalpha():   {'abc'.isalpha()}")     # True
print(f"'abc123'.isalnum(): {'abc123'.isalnum()}") # True
```

```
print(f"'abc123'.isdigit(): {'abc123'.isdigit()}") # False (есть буквы)
print(f"'   '.isspace():   {'   '.isspace()}")     # True
```

Проверки на ЕГЭ

```
# Задание №8: проверка начала/конца слова
word = "АБВГД"
if word.startswith("А"):
    print("Начинается с А")
if word.endswith("Д"):
    print("Заканчивается на Д")

# Задание №17: проверка, что строка — число
s = "12345"
if s.isdigit():
    n = int(s)
    print(f"Число: {n}")

# Задание №24: проверка регистра
if s.isupper():
    print("Все буквы заглавные")
```

Важно: `.isdigit()` и отрицательные числа

```
print(f"'-123'.isdigit(): {'-123'.isdigit()}")    # False! Минус — не
цифра
print(f"'3.14'.isdigit(): {'3.14'.isdigit()}")    # False! Точка — не
цифра

# Для проверки, можно ли преобразовать в число:
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

print(f"'-123' — число: {is_number('-123')}")
print(f"'3.14' — число: {is_number('3.14')}")
print(f"'abc' — число: {is_number('abc')}")
```

5. Методы изменения

`.replace()` — замена подстрок

```
s = "ПОПОЛАМ"

# Замена всех вхождений
print(f"Замена ПО→АП: {s.replace('ПО', 'АП')}")  # АПАПЛАМ

# Замена с ограничением количества
print(f"Только первое: {s.replace('ПО', 'АП', 1)}")  # АППОЛАМ
```

`.upper()`, `.lower()` — смена регистра

```
s = "ИнФорМатиКа"

print(f"Верхний: {s.upper()}")  # ИНФОРМАТИКА
print(f"Нижний:  {s.lower()}")  # информатика
print(f"Исходная: {s}")         # ИнФорМатиКа (строка не изменилась!)
```

`.split()` — разбиение на части

```
s = "яблоко банан груша апельсин"

# Разбиение по пробелу (по умолчанию)
words = s.split()
print(f"Слова: {words}")  # ['яблоко', 'банан', 'груша', 'апельсин']

# Разбиение по заданному разделителю
csv = "1,2,3,4,5"
numbers = csv.split(',')
print(f"Числа: {numbers}")  # ['1', '2', '3', '4', '5']

# Разбиение с ограничением
s = "раз-два-три-четыре"
parts = s.split('-', 2)
print(f"Части: {parts}")  # ['раз', 'два', 'три-четыре']
```

`.join()` — склеивание

```
words = ['Python', 'для', 'ЕГЭ']

# Склеивание через пробел
s = ' '.join(words)
print(f"С пробелами: '{s}'")  # Python для ЕГЭ

# Склеивание через другой разделитель
s = ', '.join(words)
print(f"С запятыми: '{s}'")   # Python, для, ЕГЭ

# Склеивание без разделителя
s = ''.join(words)
print(f"Слитно: '{s}'")       # PythondляЕГЭ
```

`.strip()`, `.lstrip()`, `.rstrip()` — удаление пробельных символов

```
s = "   много пробелов   "

print(f"'s.strip()':  '{s.strip()}'")   # 'много пробелов'
print(f"'s.lstrip()': '{s.lstrip()}'")  # 'много пробелов   '
print(f"'s.rstrip()': '{s.rstrip()}'")  # '   много пробелов'

# Удаление конкретных символов
s = "...точки..."
print(f"Без точек: '{s.strip('.')}'")  # 'точки'
```

6. Неизменяемость строк

Строки НЕЛЬЗЯ изменить — можно только создать новую

```
s = "Привет"

# ОШИБКА: строки неизменяемы!
# s[0] = 'п'  # TypeError: 'str' object does not support item assignment
```

Правильный способ: создать новую строку

```
s = "Привет"

# Способ 1: срезы + склеивание
s = 'п' + s[1:]  # 'привет'
print(s)

# Способ 2: replace
s = "Привет"
s = s.replace('П', 'п')
print(s)

# Способ 3: список → изменить → join
s = "Привет"
chars = list(s)     # ['П', 'р', 'и', 'в', 'е', 'т']
chars[0] = 'п'      # Меняем в списке
s = ''.join(chars)  # 'привет'
print(s)
```

Это важно для задания №12

```
# В №12 всегда создаём новую строку через replace
s = "ПОПОЛАМ"
while "ПО" in s:
    s = s.replace("ПО", "АП", 1)  # s перезаписывается новой строкой
print(s)

# Альтернатива: накапливаем в новой переменной
s = "ПОПОЛАМ"
result = ""
for char in s:
    result = result + char  # Создаётся новая строка каждый раз!
# Лучше использовать список и join (эффективнее)
```

7. Практика

Задача 1. Проверка палиндрома

Палиндром — строка, которая читается одинаково в обоих направлениях.

```
s = input("Введите строку: ")

# Убираем пробелы и приводим к нижнему регистру
```

```
s = s.lower().replace(" ", "")

# Сравниваем с перевёрнутой
if s == s[::-1]:
    print("Палиндром")
else:
    print("Не палиндром")
```

Задача 2. Подсчёт гласных в строке

```
s = input("Введите строку: ")
vowels = "аеёиоуыэюяАЕЁИОУЫЭЮЯ"

# Способ 1: цикл с проверкой
count = 0
for char in s:
    if char in vowels:
        count += 1
print(f"Гласных (способ 1): {count}")

# Способ 2: sum с генератором
count = sum(1 for char in s if char in vowels)
print(f"Гласных (способ 2): {count}")
```

Задача 3. Замена подстроки (стиль задания №12)

Дана строка. Замените все вхождения «ПО» на «АП».

```
s = input("Введите строку: ")

# Простая замена всех
s = s.replace("ПО", "АП")
print(f"Результат: {s}")
```

Задача 4. Замена до исчезновения (стиль задания №12)

Пока в строке есть «АБ», заменять «АБ» на «БА».

```
s = input("Введите строку: ")

while "АБ" in s:
    s = s.replace("АБ", "БА", 1)  # Заменяем по одному
print(f"Результат: {s}")
```

Задача 5. Двоичная строка (стиль задания №5)

Дана строка из 0 и 1. Допишите к ней справа «00» и переведите в десятичное число.

```
s = input("Введите двоичную строку: ")

# Дописываем
s = s + "00"

# Переводим в десятичное
n = int(s, 2)
print(f"Двоичная: {s}, десятичная: {n}")
```

Задача 6. Обработка слов (стиль задания №8)

Дано слово. Проверьте:

- Начинается ли с гласной?
- Заканчивается ли на согласную?
- Есть ли в нём сдвоенные буквы?

```
word = input("Введите слово: ").lower()
vowels = "аеёиоуыэюя"
consonants = "бвгджзйклмнпрстфхцчшщъь"

# Проверка начала и конца
if word and word[0] in vowels:
    print("Начинается с гласной")
if word and word[-1] in consonants:
    print("Заканчивается на согласную")

# Проверка на сдвоенные буквы
has_double = False
for i in range(len(word) - 1):
    if word[i] == word[i + 1]:
        has_double = True
        break
if has_double:
    print("Есть сдвоенные буквы")
else:
    print("Нет сдвоенных букв")
```

Домашнее задание

Задание 1. Количество слов

Дана строка, состоящая из слов, разделённых пробелами. Подсчитайте количество слов.

```
s = input("Введите строку: ")
# Ваш код здесь (подсказка: .split())
```

Задание 2. Удаление гласных

Дана строка. Удалите из неё все гласные буквы.

```
s = input("Введите строку: ")
# Ваш код здесь
```

Задание 3. Самое длинное слово

Дана строка из слов, разделённых пробелами. Найдите самое длинное слово.

```
s = input("Введите строку: ")
# Ваш код здесь
```

Задание 4. Количество цифр и букв

Дана строка. Подсчитайте отдельно количество букв и количество цифр.

```
s = input("Введите строку: ")
# Ваш код здесь (подсказка: .isalpha(), .isdigit())
```

Задание 5. Сжатие строки

Дана строка. Замените повторяющиеся подряд символы на «символ+количество». Например: «AAAABBBCC» → «A4B3C2».

```
s = input("Введите строку: ")
# Ваш код здесь
```

Что мы сегодня изучили

Тема	Ключевые моменты
Индексация	Индексы с 0, отрицательные индексы с -1
Срезы	`[start:stop:step]`, stop не включается
Поиск	`.find()`, `.rfind()`, `.count()`
Проверка	`.startswith()`, `.endswith()`, `.isdigit()`, `.isalpha()`
Изменение	`.replace()`, `.upper()`, `.lower()`, `.split()`, `.join()`
Неизменяемость	Строки нельзя изменить — только создать новую

> **Связь с ЕГЭ:** Строки — ключевой тип данных. Задачи №5, 8, 12, 14, 24 требуют уверенного владения строками.

На следующем занятии (Занятие 6)

Тема: Списки и генераторы

- Создание, индексация, срезы
- Методы: `.append()`, `.pop()`, `.sort()`, `.reverse()`
- Генераторы списков
- Решение задач №17, 25

Чек-лист занятия

- ☐ Знаю прямую и отрицательную индексацию
- ☐ Умею делать срезы `[start:stop:step]`
- ☐ Владею `.find()`, `.rfind()`, `.count()`

- ☐ Знаю методы `.replace()`, `.split()`, `.join()`
- ☐ Понимаю, что строки неизменяемы
- ☐ Выполнено домашнее задание

Занятие 6. Списки и генераторы

План занятия

1. Создание списка, индексация, срезы
2. Методы: `.append()`, `.insert()`, `.pop()`, `.remove()`, `.sort()`, `.reverse()`
3. Функции: `len()`, `min()`, `max()`, `sum()`
4. Генераторы списков (list comprehensions)
5. Вложенные генераторы
6. Практика

> **Цель занятия:** Освоить списки — основной контейнер для хранения данных. Ключевой навык для задач №17, 25, 26, 27.

1. Создание списка, индексация, срезы

Что такое список

Список (`list`) — упорядоченная изменяемая коллекция элементов. Элементы могут быть любых типов.

```
# Создание списка
empty = []                      # Пустой список
numbers = [1, 2, 3, 4, 5]       # Список чисел
mixed = [1, "два", 3.0, True]   # Смешанный список
nested = [[1, 2], [3, 4]]       # Вложенный список (матрица)

print(f"Пустой: {empty}")
print(f"Числа: {numbers}")
print(f"Смешанный: {mixed}")
print(f"Вложенный: {nested}")

# Создание из range
numbers = list(range(10))     # [0, 1, 2, ..., 9]
print(f"Из range: {numbers}")

# Создание из строки
chars = list("Python")        # ['P', 'y', 't', 'h', 'o', 'n']
print(f"Из строки: {chars}")
```

Индексация и срезы — точно как у строк

```
nums = [10, 20, 30, 40, 50]

print(f"nums[0]    = {nums[0]}")      # 10 — первый
print(f"nums[-1]   = {nums[-1]}")     # 50 — последний
print(f"nums[1:3]  = {nums[1:3]}")    # [20, 30] — срез
print(f"nums[::2]  = {nums[::2]}")    # [10, 30, 50] — с шагом
print(f"nums[::-1] = {nums[::-1]}")   # [50, 40, 30, 20, 10] — обратный

# В отличие от строк, список МОЖНО изменить по индексу
nums[0] = 100
print(f"После nums[0] = 100: {nums}")  # [100, 20, 30, 40, 50]
```

Выход за границы — ошибка

```
nums = [10, 20, 30]
# print(nums[3])   # IndexError! Индексы: 0, 1, 2
# print(nums[-4])  # IndexError! Отрицательные: -3, -2, -1
```

2. Методы списков

`.append(x)` — добавить элемент в конец

```
nums = [1, 2, 3]
nums.append(4)
print(f"После append(4): {nums}")  # [1, 2, 3, 4]

# На ЕГЭ: накопление подходящих чисел (задание №17, 25)
good_nums = []
for x in range(1, 20):
    if x % 3 == 0:
        good_nums.append(x)
print(f"Кратные 3: {good_nums}")  # [3, 6, 9, 12, 15, 18]
```

`.insert(i, x)` — вставить на позицию i

```
nums = [1, 2, 4, 5]
nums.insert(2, 3)  # Вставить 3 на индекс 2
print(f"После insert(2, 3): {nums}")  # [1, 2, 3, 4, 5]

nums.insert(0, 0)  # Вставить в начало
print(f"После insert(0, 0): {nums}")  # [0, 1, 2, 3, 4, 5]
```

`.pop(i)` — удалить и вернуть элемент по индексу

```
nums = [10, 20, 30, 40]

last = nums.pop()    # Без индекса — удаляет последний
print(f"Удалён последний: {last}, осталось: {nums}")  # 40, [10, 20, 30]

second = nums.pop(1) # Удалить по индексу 1
print(f"Удалён nums[1]: {second}, осталось: {nums}")  # 20, [10, 30]
```

`.remove(x)` — удалить первое вхождение значения

```
nums = [10, 20, 30, 20, 40]
nums.remove(20)  # Удаляет ПЕРВОЕ вхождение 20
print(f"После remove(20): {nums}")  # [10, 30, 20, 40]

# Если элемента нет — ошибка!
# nums.remove(99)  # ValueError!
```

`.sort()` и `.reverse()` — сортировка и переворот

```
nums = [5, 2, 8, 1, 9]

# Сортировка по возрастанию (на месте)
nums.sort()
print(f"По возрастанию: {nums}")  # [1, 2, 5, 8, 9]

# Сортировка по убыванию
nums.sort(reverse=True)
print(f"По убыванию: {nums}")     # [9, 8, 5, 2, 1]

# Переворот списка
nums.reverse()
print(f"После reverse: {nums}")   # [1, 2, 5, 8, 9]

# Сортировка с ключом (полезно для №26)
words = ["яблоко", "груша", "апельсин", "банан"]
words.sort(key=len)  # Сортировка по длине
print(f"По длине: {words}")

pairs = [(1, 3), (2, 1), (1, 2)]
pairs.sort()  # Сначала по первому, потом по второму
print(f"Пары: {pairs}")  # [(1, 2), (1, 3), (2, 1)]
```

3. Функции `len()`, `min()`, `max()`, `sum()`

```
nums = [7, 2, 9, 4, 1, 6]

print(f"Длина:      {len(nums)}")   # 6
print(f"Минимум:    {min(nums)}")   # 1
print(f"Максимум:   {max(nums)}")   # 9
print(f"Сумма:      {sum(nums)}")   # 29

# Работает и со строками (лексикографически)
words = ["яблоко", "груша", "апельсин"]
print(f"Мин. слово: {min(words)}")  # апельсин (по алфавиту)
print(f"Макс. слово: {max(words)}") # яблоко
```

На ЕГЭ

```
# Задание №17: анализ данных из файла
data = [12, 45, 3, 78, 22, 56]
print(f"Количество: {len(data)}")
```

```python
print(f"Максимум: {max(data)}")
print(f"Сумма: {sum(data)}")

# Подсчёт среднего (если нужно)
avg = sum(data) / len(data)
print(f"Среднее: {avg:.2f}")
```

Осторожно с пустым списком

```python
empty = []
# print(min(empty))  # ValueError! Пустой список
# print(max(empty))  # ValueError!

# Правильно: проверка на пустоту
if empty:
    print(min(empty))
else:
    print("Список пуст")
```

4. Генераторы списков (list comprehensions)

Синтаксис

```
[выражение for переменная in коллекция if условие]
```

Генератор создаёт новый список, не изменяя исходный.

```python
# Без генератора (длинно)
squares = []
for i in range(1, 11):
    squares.append(i ** 2)
print(f"Квадраты (цикл): {squares}")

# С генератором (компактно)
squares = [i ** 2 for i in range(1, 11)]
print(f"Квадраты (генератор): {squares}")
```

Генератор с фильтрацией

```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Только чётные
evens = [x for x in numbers if x % 2 == 0]
print(f"Чётные: {evens}")

# Квадраты нечётных
odd_squares = [x**2 for x in numbers if x % 2 != 0]
print(f"Квадраты нечётных: {odd_squares}")
```

Генератор с преобразованием

```python
# Длины слов
words = ["Python", "для", "ЕГЭ", "2024"]
lengths = [len(w) for w in words]
print(f"Длины: {lengths}")
```

```python
# Преобразование типов
str_nums = ["1", "2", "3", "4", "5"]
int_nums = [int(x) for x in str_nums]
print(f"Числа: {int_nums}")

# Строковые операции
names = ["  Анна  ", "  Борис  ", "  Вера  "]
clean = [name.strip() for name in names]
print(f"Очищенные: {clean}")
```

Генераторы на ЕГЭ

```python
# Чтение файла в список чисел (задание №17)
# with open("17.txt") as f:
#     data = [int(line) for line in f]

# Фильтрация данных
data = [12, 45, 3, 78, 22, 56]
filtered = [x for x in data if x % 2 == 0 and x > 20]
print(f"Чётные > 20: {filtered}")

# Преобразование для вывода
nums = [1, 2, 3, 4, 5]
output = ", ".join([str(x) for x in nums])
print(f"Вывод: {output}")
```

5. Вложенные генераторы

Генератор внутри генератора

```python
# Таблица умножения: все произведения
products = [i * j for i in range(1, 4) for j in range(1, 4)]
print(f"Произведения: {products}")  # [1, 2, 3, 2, 4, 6, 3, 6, 9]

# Эквивалентно:
products = []
for i in range(1, 4):
    for j in range(1, 4):
        products.append(i * j)
```

Вложенный генератор для матриц

```python
# Создание матрицы 3×3
matrix = [[i * 3 + j + 1 for j in range(3)] for i in range(3)]
print("Матрица 3×3:")
for row in matrix:
    print(row)
# [1, 2, 3]
# [4, 5, 6]
# [7, 8, 9]

# Транспонирование матрицы
transposed = [[row[i] for row in matrix] for i in range(3)]
```

```
print("Транспонированная:")
for row in transposed:
    print(row)
```

Вложенные генераторы на ЕГЭ (задание №8)

```
# Генерация всех трёхбуквенных слов из А, Б, В
alphabet = "АБВ"
words = [a + b + c for a in alphabet for b in alphabet for c in alphabet]
print(f"Всего слов: {len(words)}")  # 27

# С фильтрацией: без "АА"
words_filtered = [a + b + c for a in alphabet
                            for b in alphabet
                            for c in alphabet
                            if a + b + c ≠ "ААА" and "АА" not in a + b +
c]
print(f"Слов без 'АА': {len(words_filtered)}")
```

Осторожно: читаемость

```
# ПЛОХО: слишком сложный генератор
# result = [x for x in range(100) if x % 2 == 0 if x % 3 == 0 if x > 10]

# ХОРОШО: разбить на несколько шагов или использовать цикл
result = []
for x in range(100):
    if x % 2 == 0 and x % 3 == 0 and x > 10:
        result.append(x)
```

> **Правило:** Если генератор не помещается в одну строку (~80 символов) или содержит больше 2 `for` / `if` — лучше написать обычный цикл.

6. Практика

Задача 1. Фильтрация списка (оставить только чётные)

```
numbers = [3, 7, 2, 8, 1, 6, 9, 4]

# Способ 1: цикл
evens = []
for x in numbers:
    if x % 2 == 0:
        evens.append(x)
print(f"Чётные (цикл): {evens}")

# Способ 2: генератор
evens = [x for x in numbers if x % 2 == 0]
print(f"Чётные (генератор): {evens}")
```

Задача 2. Сортировка по убыванию

```
numbers = [3, 7, 2, 8, 1, 6, 9, 4]
```

```
# Сортировка на месте
numbers.sort(reverse=True)
print(f"По убыванию: {numbers}")

# Без изменения исходного: sorted()
numbers = [3, 7, 2, 8, 1, 6, 9, 4]
sorted_nums = sorted(numbers, reverse=True)
print(f"Исходный: {numbers}")
print(f"Отсортированный: {sorted_nums}")
```

Задача 3. Создание списка квадратов

```
# Квадраты чисел от 1 до 10
squares = [x**2 for x in range(1, 11)]
print(f"Квадраты: {squares}")

# Квадраты только чётных
even_squares = [x**2 for x in range(1, 11) if x % 2 == 0]
print(f"Квадраты чётных: {even_squares}")
```

Задача 4. Анализ данных (стиль задания №17)

Дан список чисел. Найдите количество, сумму и максимум среди элементов, кратных 3.

```
data = [12, 7, 9, 15, 22, 18, 5, 21]

# Фильтрация
filtered = [x for x in data if x % 3 == 0]
print(f"Кратные 3: {filtered}")
print(f"Количество: {len(filtered)}")
print(f"Сумма: {sum(filtered)}")
print(f"Максимум: {max(filtered)}")
```

Задача 5. Сортировка с ключом (стиль задания №26)

Дан список товаров (название, цена). Отсортируйте по цене.

```
goods = [("Ручка", 30), ("Тетрадь", 45), ("Карандаш", 15), ("Ластик", 25)]

# Сортировка по цене (второй элемент кортежа)
goods.sort(key=lambda x: x[1])
print("По цене:")
for name, price in goods:
    print(f"  {name}: {price} руб.")

# Сортировка по названию
goods.sort(key=lambda x: x[0])
print("По названию:")
for name, price in goods:
    print(f"  {name}: {price} руб.")
```

Задача 6. Удаление дубликатов

Дан список с повторяющимися элементами. Создайте список без дубликатов.

```
numbers = [1, 3, 2, 3, 4, 1, 5, 2, 4]
```

```python
# Способ 1: цикл с проверкой
unique = []
for x in numbers:
    if x not in unique:
        unique.append(x)
print(f"Без дубликатов: {unique}")

# Способ 2: через set (не сохраняет порядок)
unique_set = list(set(numbers))
print(f"Через set: {unique_set}")
```

Домашнее задание

Задание 1. Список делителей

Дано число N. Создайте список всех его делителей.

```python
n = int(input("Введите число: "))
# Ваш код здесь
```

Задание 2. Пересечение списков

Даны два списка. Найдите их пересечение (элементы, которые есть в обоих).

```python
list1 = [1, 2, 3, 4, 5]
list2 = [3, 5, 7, 9]
# Ваш код здесь
```

Задание 3. Генератор: числа, кратные 5

Создайте список чисел от 1 до 100, кратных 5, с помощью генератора.

```python
# Ваш код здесь: [x for x in range( ... ) if  ... ]
```

Задание 4. Сортировка по длине слова

Дан список слов. Отсортируйте его по длине слова (от коротких к длинным).

```python
words = ["Python", "для", "ЕГЭ", "информатика", "код"]
# Ваш код здесь
```

Задание 5. Матрица: сумма строк

Дана матрица 3×3 (список списков). Найдите сумму элементов каждой строки.

```python
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
# Ваш код здесь
```

Что мы сегодня изучили

Тема	Ключевые моменты
Создание списка	`[]`, `list(range())`, `list("строка")`
Индексация и срезы	Как у строк, но списки изменяемы
Методы	`.append()`, `.insert()`, `.pop()`, `.remove()`, `.sort()`, `.reverse()`
Функции	`len()`, `min()`, `max()`, `sum()`
Генераторы	`[x for x in range if условие]`
Вложенные генераторы	Для матриц, комбинаций, фильтрации

> **Связь с ЕГЭ:** Списки — основной контейнер для данных. Задачи №17, 25, 26, 27 активно используют списки и генераторы.

На следующем занятии (Занятие 6.5)

Тема: Анализ простого алгоритма (задание №6)

- • Трассировка алгоритмов
- • Python-брутфорс для проверки
- • Типичные ловушки

Чек-лист занятия

- ☐ Умею создавать список разными способами
- ☐ Знаю методы `.append()`, `.pop()`, `.sort()`, `.reverse()`
- ☐ Понимаю разницу между `.sort()` (на месте) и `sorted()` (новый)
- ☐ Владею генераторами списков с условием
- ☐ Могу написать вложенный генератор
- ☐ Выполнено домашнее задание

Занятие 6.5. Задание №6 — Исполнитель Черепаха

План занятия

1. Знакомство с модулем `turtle` — все команды

2. Холст и как на нём разместить координатную сетку
3. Переписывание алгоритма из условия на Python
4. Тип А: площадь фигуры через заливку
5. Тип Б: повторные посещения — рисуем и смотрим глазами

> **Цель:** Научиться визуализировать алгоритм Черепахи и анализировать результат вручную, глядя на рисунок.

1. Модуль `turtle` — все команды

Импорт и создание Черепахи

```
from turtle import *

# Создаём Черепаху (автоматически)
# Начальное положение: (0, 0), голова направлена ВПРАВО (на восток)
```

Главное отличие от ЕГЭ

- В **ЕГЭ** Черепаха изначально смотрит **ВВЕРХ** (вдоль OY).
- В **`turtle`** Черепаха изначально смотрит **ВПРАВО** (вдоль OX).

Поэтому **первой командой всегда** даём:

```
left(90)   # Поворот на 90° против часовой → теперь смотрит ВВЕРХ
```

Таблица команд

Команда ЕГЭ	Команда Python (краткая)	Команда Python (полная)	Что делает
`Вперёд n`	`fd(n)`	`forward(n)`	Движение вперёд на `n`
`Назад n`	`bk(n)`	`backward(n)`	Движение назад на `n`
`Направо m`	`rt(m)`	`right(m)`	Поворот по часовой стрелке на `m` градусов
`Налево m`	`lt(m)`	`left(m)`	Поворот против часовой стрелки на `m` градусов
`Поднять хвост`	`up()`	`penup()`	Перестать рисовать
`Опустить хвост`	`pd()`	`pendown()`	Начать рисовать

Цикл `Повтори k [команды]`

```
Повтори 4 [Вперёд 10 Направо 90]
```

Превращается в:

```
for _ in range(4):
    fd(10)
    rt(90)
```

Настройки вида

```
speed(0)          # Максимальная скорость (без анимации)
hideturtle()      # Скрыть значок Черепахи (мешает смотреть)
pensize(2)        # Толщина линии (по желанию)
```

2. Холст и координатная сетка

Что такое холст

Холст (`canvas`) — это область, на которой Черепаха рисует. Когда мы открываем окно `turtle`, создаётся холст размером примерно 960×810 пикселей с центром в точке `(0, 0)`.

Координаты на холсте

- Центр холста: `(0, 0)`.
- Ось X: вправо `+`, влево `-`.
- Ось Y: вверх `+`, вниз `-`.
- Единица измерения — **пиксели**.

Как нанести координатную сетку

Чтобы видеть целочисленные точки, мы рисуем **сетку с шагом 1**. Но точки будут мелкими. Поэтому используют **масштаб** `k` : все расстояния умножают на 10 или 20. Тогда 1 единица = 10 (или 20) пикселей.

```
k = 20  # Масштаб: 1 шаг Черепахи = 20 пикселей
```

Как нарисовать сетку (для визуального анализа):

```
from turtle import *

k = 20
speed(0)
hideturtle()

# Рисуем точки на целых координатах
for x in range(-10, 11):
    for y in range(-10, 11):
        up()
        goto(x * k, y * k)
        pd()
        dot(3, 'gray')  # Серая точка диаметром 3 пикселя
```

Как прочитать координаты точки с рисунка

- Найдите точку на рисунке.
- Поделите её координаты в пикселях на `k`.

• Например, точка на рисунке в `(60, 40)` при `k=20` → реальные координаты `(3, 2)`.

> **На экзамене сетку рисовать не обязательно.** Достаточно нарисовать фигуру и визуально оценить, какие целые точки попали внутрь / на границу / были посещены несколько раз.

3. Переписывание алгоритма на Python

Правила перевода

Конструкция ЕГЭ	Python
`Повтори k [Команды]`	`for _ in range(k):` + отступ
`Вперёд n`	`fd(n * k)`
`Направо m`	`rt(m)`
`Налево m`	`lt(m)`
`Поднять хвост`	`up()`
`Опустить хвост`	`pd()`

Пример 1. Квадрат

```
Повтори 4 [Вперёд 5 Направо 90]
```

```
for _ in range(4):
    fd(5 * k)
    rt(90)
```

Пример 2. С поднятием хвоста

```
Повтори 2 [Вперёд 6 Направо 90 Вперёд 4 Направо 90]
Поднять хвост
Вперёд 2
Опустить хвост
Повтори 2 [Вперёд 3 Направо 90 Вперёд 2 Направо 90]
```

```
# Первый прямоугольник
for _ in range(2):
    fd(6 * k)
    rt(90)
    fd(4 * k)
    rt(90)

# Перемещение без рисования
up()
fd(2 * k)
pd()

# Второй прямоугольник
for _ in range(2):
    fd(3 * k)
    rt(90)
```

```
    fd(2 * k)
    rt(90)
```

Пример 3. Треугольник

```
Повтори 3 [Вперёд 7 Направо 120]
```

```
for _ in range(3):
    fd(7 * k)
    rt(120)
```

Пример 4. Вложенные циклы

```
Повтори 4 [Повтори 4 [Вперёд 6 Направо 90] Вперёд 10 Направо 90 Вперёд 3]
```

```
for _ in range(4):           # Внешний цикл
    for _ in range(4):       # Внутренний цикл (квадрат)
        fd(6 * k)
        rt(90)
    fd(10 * k)               # После квадрата
    rt(90)
    fd(3 * k)
```

4. Тип А: площадь фигуры через заливку

Идея

1. Рисуем замкнутую фигуру.
2. Заливаем её цветом (`begin_fill` / `end_fill`).
3. Сканируем холст и считаем пиксели залитого цвета.

Как отделить границу от внутренности

```
color('black', 'red')  # Линия — чёрная, заливка — красная
```

- Точки на линии → чёрные.
- Точки внутри → красные.

Полный шаблон для Типа А

```
from turtle import *

k = 10                          # Масштаб
speed(0); hideturtle()          # Без анимации, без значка
color('black', 'red')           # Линия чёрная, заливка красная

# Рисуем и заливаем фигуру
begin_fill()
#  ... здесь алгоритм из условия  ...
end_fill()

# Сканируем холст
sc = getcanvas()
```

```
# Только внутри (без границы) — красные точки
inside = sum(1 for x in range(-300*k, 300*k, k)
             for y in range(-300*k, 300*k, k)
             if (h := sc.find_overlapping(x, y, x, y))
             and sc.itemcget(h[-1], 'fill') == 'red')

# Внутри + граница — красные или чёрные
total = sum(1 for x in range(-300*k, 300*k, k)
            for y in range(-300*k, 300*k, k)
            if (h := sc.find_overlapping(x, y, x, y))
            and sc.itemcget(h[-1], 'fill') in ('red', 'black'))

print(f"Внутри (без границы): {inside}")
print(f"Внутри + граница: {total}")
done()
```

Разбор ключевых строк

`sc = getcanvas()` — получаем объект холста. Холст хранит всё, что нарисовано.

`sc.find_overlapping(x1, y1, x2, y2)` — возвращает **список ID объектов**, которые пересекаются с прямоугольником `(x1, y1) – (x2, y2)`. Если `x1 == x2` и `y1 == y2`, то это **точка**.

```
h = sc.find_overlapping(50, 30, 50, 30)
# h = [3, 7] — объекты с ID 3 и 7 находятся в точке (50, 30)
```

`sc.itemcget(id, 'fill')` — возвращает **цвет заливки** объекта с указанным ID.

```
color = sc.itemcget(7, 'fill')
# color = 'red' — объект залит красным
```

`h[-1]` — последний элемент списка. Это **верхний** объект (нарисованный последним).

`sum(1 for ... if ...)` — подсчёт количества точек, удовлетворяющих условию.

5. Тип Б: повторные посещения — рисуем и смотрим

Идея

В задачах на **повторные посещения** не нужна заливка и не нужен код для подсчёта. Мы:

1. **Рисуем траекторию** Черепахи (линии).
2. **Наносим точки** с целочисленными координатами поверх рисунка.
3. **Смотрим глазами**, какие точки накрыты линией **более одного раза**.

Код для визуализации траектории с точками

```
from turtle import *

k = 20  # Масштаб
speed(0); hideturtle()
```

```
# 1. Рисуем сетку из целочисленных точек
for x in range(-15, 16):
    for y in range(-15, 16):
        up()
        goto(x * k, y * k)
        pd()
        dot(4, 'lightgray')  # Светло-серая точка

# 2. Рисуем траекторию Черепахи
goto(0, 0)           # Возвращаемся в начало
left(90)             # Направляем вверх (как в ЕГЭ)
pencolor('black')    # Цвет линии — чёрный
pensize(2)           # Толщина линии
pd()                 # Хвост опущен

# ── Алгоритм из условия ──
# (здесь пишем команды Черепахи)
# ── Конец алгоритма ──

done()
```

Как анализировать

1. Запускаем код. Открывается окно с рисунком.
2. Серые точки — это **все целочисленные координаты**.
3. Чёрная линия — **траектория Черепахи**.
4. Смотрим на точки, через которые линия прошла **два или более раз**.
5. Считаем их **вручную**.

> **Совет:** Если линия толстая (`pensize(3)`), она может закрывать соседние точки. Уменьшите толщину до 1 или увеличьте масштаб `k` до 30.

6. Разбор конкретных задач

Задача 1. Два прямоугольника (Тип А)

Условие:

```
Повтори 2 [Вперёд 27 Направо 90 Вперёд 8 Направо 90]
Поднять хвост
Вперёд 4 Направо 90 Вперёд 2 Налево 90
Опустить хвост
Повтори 2 [Вперёд 17 Направо 90 Вперёд 7 Направо 90]
```

Нужно: количество целых точек внутри объединения фигур, **включая точки на линиях**.

Код:

```
from turtle import *

k = 10; speed(0); color('black', 'red'); hideturtle()
```

```
# Первый прямоугольник 27×8
begin_fill()
[(fd(27*k), rt(90), fd(8*k), rt(90)) for _ in range(2)]
end_fill()

# Перемещение
up()
fd(4*k); rt(90); fd(2*k); lt(90)
pd()

# Второй прямоугольник 17×7
begin_fill()
[(fd(17*k), rt(90), fd(7*k), rt(90)) for _ in range(2)]
end_fill()

# Подсчёт: внутри + граница
sc = getcanvas()
total = sum(1 for x in range(-300*k, 300*k, k)
            for y in range(-300*k, 300*k, k)
            if (h := sc.find_overlapping(x, y, x, y))
            and sc.itemcget(h[-1], 'fill') in ('red', 'black'))
print(total)
done()
```

Что происходит:

1. Рисуем первый прямоугольник 27×8 и заливаем красным. Линия чёрная.
2. Перемещаемся без рисования.
3. Рисуем второй прямоугольник 17×7 и заливаем красным.
4. Сканируем холст с шагом `k=10`. Каждая точка `(x, y)` — это целая координата `(x/10, y/10)`.
5. Если в точке есть красный или чёрный объект — значит, она принадлежит фигуре (внутри или на границе).
6. Считаем все такие точки.

Задача 2. Треугольник без границы (Тип А)

Условие:

```
Повтори 3 [Вперёд 7 Направо 120]
```

Нужно: количество целых точек внутри, **точки на линии не учитывать**.

Код:

```
from turtle import *

k = 10; speed(0); color('black', 'red'); hideturtle()

begin_fill()
[(fd(7*k), rt(120)) for _ in range(3)]
end_fill()
```

```
sc = getcanvas()
inside = sum(1 for x in range(-300*k, 300*k, k)
             for y in range(-300*k, 300*k, k)
             if (h := sc.find_overlapping(x, y, x, y))
             and sc.itemcget(h[-1], 'fill') == 'red')
print(inside)
done()
```

Отличие от задачи 1:

- Считаем только `'red'`, не `'black'`. Чёрные точки (граница) исключаются.

Задача 3. Повторные посещения (Тип Б) — визуальный анализ

Условие:

```
Повтори 4 [
    Повтори 4 [Вперёд 6 Направо 90]
    Вперёд 10
    Направо 90
    Вперёд 3
]
```

Нужно: количество точек с целочисленными координатами, где Черепаха **побывала более одного раза**.

Шаг 1. Рисуем траекторию и сетку.

```
from turtle import *

k = 20; speed(0); hideturtle()

# Рисуем целочисленные точки (сетку)
for x in range(-10, 20):
    for y in range(-10, 20):
        up()
        goto(x * k, y * k)
        pd()
        dot(4, 'lightgray')

# Рисуем траекторию
goto(0, 0); left(90)
pencolor('black'); pensize(2); pd()

for _ in range(4):
    for _ in range(4):
        fd(6 * k)
        rt(90)
    fd(10 * k)
    rt(90)
    fd(3 * k)

done()
```

Шаг 2. Анализируем рисунок глазами.

- Серые точки — целые координаты.

- Чёрная линия — траектория.
- Ищем серые точки, через которые линия прошла **2 и более раз**.

Что мы видим на рисунке:

1. Внутренний цикл 4 раза рисует квадрат 6×6. После 4 повторений (`fd(6), rt(90)`) Черепаха возвращается в ту же точку. Значит, **каждая сторона квадрата проходится 4 раза** (по одному на каждый внешний цикл).
2. Точки на сторонах квадрата 6×6 посещаются **4 раза** (один раз за каждый внешний цикл).
3. Отрезки `fd(10)` и `fd(3)` проходятся **1 раз** за каждый внешний цикл, но они не накладываются друг на друга (разные внешние циклы смещают Черепаху).

Считаем вручную:

- Квадрат 6×6: на каждой стороне 7 точек (включая углы). Но углы считаются дважды. Всего уникальных точек на квадрате: 4 × 6 = 24 (без учёта повторного счёта углов).
- Эти 24 точки посещены 4 раза каждая → это **24 точки**, посещённые более одного раза.
- Дополнительно: проверяем, не накладываются ли `fd(10)` и `fd(3)` из разных циклов. На рисунке видно, что они расходятся в разные стороны и не пересекаются.

Ответ: 24 (или другое число в зависимости от конкретных длин).

> **Главный навык:** Не написать код для автоподсчёта, а **понять по рисунку**, какие участки траектории накладываются друг на друга.

Сводная таблица: Тип А vs Тип Б

Критерий	Тип А (площадь)	Тип Б (повторные посещения)
Вопрос	«точек внутри области»	«побывала более одного раза»
Метод	Заливка + сканирование холста	Рисуем траекторию + сетку, смотрим глазами
Код	`begin_fill()` / `end_fill()` + `getcanvas()`	`goto()` для сетки + команды Черепахи
Автоматизация	Да (код считает сам)	Нет (человек считает по рисунку)

Домашнее задание

Задание 1 (Тип А)

```
Повтори 4 [Вперёд 10 Направо 90]
```

Запустите код с заливкой. Сколько точек внутри + граница? Сколько только внутри?

Задание 2 (Тип Б)

```
Повтори 2 [Вперёд 5 Направо 90 Вперёд 3 Направо 90]
```

Нарисуйте траекторию с сеткой. Сколько точек посещено более одного раза? Посчитайте глазами.

Задание 3 (Тип Б, сложнее)

- `Повтори 3 [Вперёд 4 Направо 120]`

Нарисуйте траекторию. Сколько точек посещено более одного раза? (Подсказка: треугольник, 3 прохода по одному и тому же пути.)

Чек-лист занятия

- ☐ Знаю все команды `turtle` и их краткие имена
- ☐ Умею переписывать `Повтори` в `for _ in range()`
- ☐ Понимаю, зачем нужен масштаб `k`
- ☐ Умею рисовать координатную сетку (`dot()`)
- ☐ Знаю, как работает заливка и сканирование холста
- ☐ Умею визуально определять повторные посещения точек

Занятие 7. Функции

План занятия

1. Зачем нужны функции
2. `def` и `return` — базовый синтаксис
3. Аргументы по умолчанию
4. Области видимости: локальные и глобальные переменные
5. Функции от нескольких переменных
6. Лямбда-функции
7. Практика: оформление алгоритмов №5 в функции

> **Цель занятия:** Научиться оформлять алгоритмы в виде функций. Это главный инструмент для задач №5, 14, 16, 19–21, 23.

1. Зачем нужны функции

Проблема без функций

В заданиях №5, 14, 16, 19–21, 23 часто нужно **много раз выполнить один и тот же алгоритм** для разных входных данных. Без функций приходится копировать код или писать громоздкие циклы.

Пример (№5): Алгоритм обрабатывает число N. Нужно перебрать N от 1 до 1000 и найти то, которое даст нужный результат.

```
# Без функции — код дублируется или запутывается
for N in range(1, 1001):
    # ... 10 строк алгоритма ...
    if result == target:
        print(N)
```

С функцией

```
def algorithm(N):
    # ... 10 строк алгоритма ...
    return result

for N in range(1, 1001):
    if algorithm(N) == target:
        print(N)
```

Преимущества:

- Код читаемый: понятно, что делает `algorithm(N)`.
- Легко тестировать: можно вызвать функцию для одного числа и проверить.
- Нет дублирования: алгоритм описан один раз.

2. `def` и `return` — базовый синтаксис

Структура функции

```
def имя_функции(аргумент1, аргумент2):
    """Описание (необязательно)."""
    # Тело функции — вычисления
    результат = ...
    return результат  # Возвращаем результат
```

Простой пример

```
def add(a, b):
    """Возвращает сумму двух чисел."""
    return a + b

result = add(3, 5)
print(result)  # 8
```

`return` vs `print`

- `return` — **возвращает значение** из функции. Его можно присвоить переменной, использовать в выражении.
- `print` — просто **выводит на экран**. Не возвращает значение.

```
def bad_sum(a, b):
    print(a + b)  # Только выводит, не возвращает!
```

```python
def good_sum(a, b):
    return a + b  # Возвращает значение

x = bad_sum(3, 5)   # x = None (функция ничего не вернула)
y = good_sum(3, 5)  # y = 8
```

Несколько `return`

Функция может иметь несколько `return`. Как только выполняется `return`, функция завершается.

```python
def sign(x):
    if x > 0:
        return 1
    elif x < 0:
        return -1
    else:
        return 0

print(sign(10))   # 1
print(sign(-5))   # -1
print(sign(0))    # 0
```

3. Аргументы по умолчанию

Синтаксис

```python
def имя(аргумент=значение_по_умолчанию):
    ...
```

Если аргумент не передан, используется значение по умолчанию.

```python
def greet(name="мир"):
    print(f"Привет, {name}!")

greet("Анна")  # Привет, Анна!
greet()        # Привет, мир! (использовано значение по умолчанию)
```

На ЕГЭ: диапазон перебора

```python
def find_answer(start=1, end=1000):
    """Ищет ответ в диапазоне [start, end]."""
    for N in range(start, end + 1):
        if algorithm(N) == target:
            return N
    return None  # Не найдено
```

Опасность: изменяемые значения по умолчанию

```python
# НЕ ДЕЛАЙТЕ ТАК:
def bad(data=[]):  # Список создаётся ОДИН раз при определении функции!
    data.append(1)
    return data

print(bad())  # [1]
```

```
print(bad())  # [1, 1] — неожиданно!

# ПРАВИЛЬНО:
def good(data=None):
    if data is None:
        data = []
    data.append(1)
    return data
```

> **На ЕГЭ эта ловушка встречается редко, но знать о ней нужно.**

4. Области видимости: локальные и глобальные переменные

Локальные переменные

Переменные, созданные **внутри функции**, видны только внутри этой функции.

```
def my_func():
    x = 10  # Локальная переменная
    print(x)

my_func()
# print(x)  # ОШИБКА! x не существует вне функции
```

Глобальные переменные

Переменные, созданные **вне всех функций**, видны везде (но только для чтения).

```
x = 10  # Глобальная переменная

def show():
    print(x)  # Можно читать глобальную переменную

show()  # 10
```

Изменение глобальной переменной

Если внутри функции присвоить значение переменной, Python создаст **новую локальную** переменную, а не изменит глобальную.

```
x = 10

def change():
    x = 20  # Создаётся НОВАЯ локальная x, глобальная не меняется!
    print(f"Внутри: {x}")

change()           # Внутри: 20
print(f"Снаружи: {x}")  # Снаружи: 10 (не изменилась!)
```

`global` — изменение глобальной переменной

```
x = 10

def change():
```

```
    global x  # Говорим Python: «x — глобальная переменная»
    x = 20

change()
print(x)  # 20 (изменилась!)
```

> **На ЕГЭ `global` используется редко.** Обычно достаточно передавать значения через аргументы и возвращать через `return`.

5. Функции от нескольких переменных

Базовый синтаксис

```
def func(a, b, c):
    return a + b + c
```

Передача нескольких значений

```
# Позиционные аргументы
def rectangle_area(width, height):
    return width * height

print(rectangle_area(10, 5))      # 50
print(rectangle_area(height=5, width=10))  # 50 (именованные)
```

На ЕГЭ: алгоритм с несколькими параметрами

```
def algorithm(N, base=2):
    """Обрабатывает число N в системе счисления с основанием base."""
    # Перевод в base-систему
    digits = []
    temp = N
    while temp > 0:
        digits.append(temp % base)
        temp //= base
    # ... обработка ...
    return result
```

Возврат нескольких значений

Функция может вернуть несколько значений через запятую. Фактически возвращается **кортеж**.

```
def min_max(a, b):
    if a < b:
        return a, b
    else:
        return b, a

mn, mx = min_max(7, 3)
print(f"min = {mn}, max = {mx}")  # min = 3, max = 7
```

6. Лямбда-функции

Что это

Лямбда-функция — это **короткая анонимная функция** в одну строку.

```
lambda аргументы: выражение
```

```
# Обычная функция
def square(x):
    return x ** 2

# Лямбда-функция (то же самое)
square_lambda = lambda x: x ** 2

print(square(5))         # 25
print(square_lambda(5))  # 25
```

Где используются на ЕГЭ

1. Сортировка с ключом (№26):

```
pairs = [(1, 5), (3, 2), (2, 8)]
pairs.sort(key=lambda x: x[1])  # Сортировка по второму элементу
print(pairs)  # [(3, 2), (1, 5), (2, 8)]
```

2. `map()` и `filter()` (№17, 25):

```
nums = [1, 2, 3, 4, 5]

# map: применить функцию ко всем элементам
squares = list(map(lambda x: x**2, nums))
print(squares)  # [1, 4, 9, 16, 25]

# filter: оставить только подходящие
evens = list(filter(lambda x: x % 2 == 0, nums))
print(evens)  # [2, 4]
```

> **Лямбды необязательны.** Всё то же самое можно сделать через обычные функции или генераторы списков. Но с лямбдами код короче.

7. Практика: оформление алгоритмов №5 в функции

Задача 1 (8-битная инверсия)

Условие:

Автомат обрабатывает натуральное число N ($128 \le N \le 255$):

1. Строится 8-битная двоичная запись числа N.
2. Все цифры заменяются на противоположные ($0 \to 1$, $1 \to 0$).
3. Полученное число переводится в десятичную запись.

4. Из исходного числа вычитается полученное, разность выводится на экран.

Вопрос: Какое число N даст результат 105?

Оформляем алгоритм в функцию:

```
def algorithm(N):
    """Возвращает результат работы алгоритма для числа N."""
    # 1. 8-битная двоичная запись
    binary = bin(N)[2:].zfill(8)  # zfill(8) — дополнить нулями до 8 символов

    # 2. Инвертируем каждый бит
    inverted = ''.join('1' if b == '0' else '0' for b in binary)

    # 3. Переводим в десятичную
    inverted_num = int(inverted, 2)

    # 4. Разность
    return N - inverted_num

# Перебираем N
for N in range(128, 256):
    if algorithm(N) == 105:
        print(N)
        break
```

Разбор:

- `bin(N)[2:]` — двоичная запись без `0b`.
- `.zfill(8)` — дополняет нулями слева до 8 символов.
- `int(s, 2)` — перевод двоичной строки в десятичное число.
- Функция `algorithm(N)` возвращает результат. Цикл перебирает N и сравнивает с 105.

Задача 2 (троичная система с заменой)

Условие:

Алгоритм для натурального числа N:

1. Троичная запись числа N.
2. Заменяем 0→2, 2→0. Удаляем ведущие нули.
3. Переводим в десятичную.
4. Результат R = |N − полученное|.

Вопрос: Наименьшее N, при котором R = 1 864 246.

Оформляем в функцию:

```
def algorithm(N):
    """Возвращает R для числа N."""
    # 1. Троичная запись
    ternary = ''
```

```
    temp = N
    while temp > 0:
        ternary = str(temp % 3) + ternary
        temp //= 3

    # 2. Замена 0↔2
    replaced = ''.join('2' if d == '0' else ('0' if d == '2' else d) for
d in ternary)
    # Удаляем ведущие нули
    replaced = replaced.lstrip('0') or '0'

    # 3. Перевод в десятичную
    new_num = int(replaced, 3) if replaced else 0

    # 4. Модуль разности
    return abs(N - new_num)

# Ищем наименьшее N
target = 1864246
N = 1
while True:
    if algorithm(N) == target:
        print(N)
        break
    N += 1
    if N % 100000 == 0:  # Прогресс для больших переборов
        print(f"Проверено {N}...")
```

Задача 3 (двоичная с дописыванием)

Условие:

Алгоритм для натурального числа N:

1. Двоичная запись N.
2. Если сумма цифр десятичной записи N нечётна — дописать 1, иначе 0. 3–4. Повторить пункт 2 ещё два раза.
3. Результат R — десятичная запись полученного числа.

Вопрос: Наименьшее R > 2054.

Оформляем в функцию:

```
def sum_of_digits(num):
    """Сумма цифр десятичной записи числа."""
    return sum(int(d) for d in str(num))

def algorithm(N):
    """Возвращает R для числа N."""
    binary = bin(N)[2:]  # Двоичная запись
    current = N

    for _ in range(3):  # Три раза дописываем
        if sum_of_digits(current) % 2 != 0:  # Нечётная сумма
            binary += '1'
```

```
        else:
            binary += '0'
        current = int(binary, 2)  # Обновляем текущее число

    return current

# Ищем наименьшее R > 2054
results = []
for N in range(1, 500):  # Достаточно большой диапазон
    R = algorithm(N)
    if R > 2054:
        results.append(R)

print(min(results))
```

Разбор:

- Функция `sum_of_digits` — вспомогательная. Считает сумму цифр числа.
- `algorithm(N)` — основная. Возвращает R.
- Цикл собирает все R > 2054 и находит минимум.

Домашнее задание

Задание 1

Напишите функцию `to_base(n, base)`, которая переводит число `n` в систему счисления с основанием `base` (2..16). Возвращает строку.

Задание 2

Напишите функцию `is_prime(n)`, которая возвращает `True`, если число простое, и `False` иначе.

Задание 3

Оформите решение задачи №5 из открытого банка ФИПИ в виде функции. Найдите ответ перебором.

Что мы сегодня изучили

Тема	Ключевые моменты
`def` и `return`	Функция принимает аргументы и возвращает результат
`return` vs `print`	`return` — для передачи значения, `print` — для вывода
Аргументы по умолчанию	`def f(x=10)` — можно не передавать
Локальные vs глобальные	Внутри функции — локальные, снаружи — глобальные
Лямбда-функции	`lambda x: x**2` — короткая функция в одну строку
Функции для №5	Оформляем алгоритм в функцию и перебираем N

> **Связь с ЕГЭ:** Функции — фундамент. Без них невозможно решать №5, 14, 16, 19–21, 23. Это главный инструмент организации кода.

На следующем занятии (Занятие 8)

Тема: Философия брутфорса и itertools

- `itertools.product`, `permutations`
- Оценка количества итераций
- Задачи №2, 8

Чек-лист занятия

- ☐ Умею писать функцию с `def` и `return`
- ☐ Понимаю разницу между `return` и `print`
- ☐ Знаю, что такое локальные и глобальные переменные
- ☐ Могу написать лямбда-функцию
- ☐ Оформил алгоритм №5 в виде функции
- ☐ Выполнено домашнее задание

Глава 2. Брутфорс

Занятие 8. Философия брутфорса и itertools

План занятия

1. Что такое брутфорс и когда он работает
2. `itertools.product` — декартово произведение
3. `itertools.permutations` — перестановки
4. Задание №2: полный разбор
5. Важно: когда заданных строк много — используем компьютер для построения полной таблицы, остальное анализом
6. Практика

> **Цель занятия:** Освоить главный инструмент ЕГЭ — полный перебор. Научиться решать задание №2 за 3 минуты.

1. Что такое брутфорс и когда он работает

Определение

Брутфорс (от англ. *brute force* — грубая сила) — метод решения задачи путём **полного перебора всех возможных вариантов**.

Вместо того чтобы выводить сложную формулу, мы:

1. Генерируем все варианты.
2. Проверяем каждый вариант.
3. Выводим подходящий.

Когда брутфорс работает

Брутфорс работает, когда **количество вариантов обозримо** (обычно до 10^6–10^7).

Задача	Количество вариантов	Время брутфорса
Таблица истинности (4 переменные)	$2^4 = 16$ строк	Мгновенно
Перебор N от 1 до 1000 (№5)	1000	Мгновенно
Слова длины 5 из 4 букв (№8)	$4^5 = 1024$	Мгновенно
Перестановки 4 столбцов (№2)	$4! = 24$	Мгновенно
Выбор 3 строк из 16 и их порядок	$16 \times 15 \times 14 = 3360$	Мгновенно
Слова длины 10 из 15 букв (№8 сложное)	$15^{10} \approx 5{,}7 \times 10^{11}$	Неприемлемо

Философия брутфорса на ЕГЭ

> **Если можно перебрать — перебирай.** Это быстрее, чем думать, и исключает логические ошибки.

Брутфорс применяется в заданиях:

- №2 — таблицы истинности.
- №5 — поиск числа N.
- №8 — комбинаторика (простые случаи).
- №14 — поиск цифры/основания.
- №15 — поиск значения A (когда A — число).
- №25 — поиск чисел с заданными свойствами.

2. `itertools.product` — декартово произведение

Что это

`product` генерирует **все возможные комбинации** элементов из переданных наборов. Это как вложенные циклы, но в одну строку.

```
from itertools import product

# Все комбинации двух цифр
for a, b in product([0, 1], [0, 1]):
    print(a, b)
# 0 0
# 0 1
# 1 0
# 1 1
```

`repeat` — повторение одного набора

Если нужно перебрать комбинации из **одного и того же множества** несколько раз, используют `repeat`.

```
from itertools import product

# Все наборы из 3 нулей и единиц (2³ = 8 вариантов)
for x, y, z in product([0, 1], repeat=3):
    print(x, y, z)
```

Это эквивалентно трём вложенным циклам:

```
for x in [0, 1]:
    for y in [0, 1]:
        for z in [0, 1]:
            print(x, y, z)
```

На ЕГЭ: таблица истинности (№2)

```
from itertools import product

# Все 16 строк таблицы для 4 переменных
for x, y, z, w in product([0, 1], repeat=4):
    F = (x or y) and not (z and w)  # Пример выражения
    print(x, y, z, w, int(F))
```

3. `itertools.permutations` — перестановки

Что это

`permutations` генерирует **все перестановки** элементов — все способы расположить их в разном порядке.

```
from itertools import permutations

# Все перестановки трёх букв
for p in permutations('ABC'):
    print(''.join(p))
# ABC, ACB, BAC, BCA, CAB, CBA
```

Перестановки с выбором длины

```
from itertools import permutations

# Размещения из 4 элементов по 2 (порядок важен)
for p in permutations([0, 1, 2, 3], 2):
    print(p)
# (0,1), (0,2), (0,3), (1,0), (1,2), (1,3), (2,0), (2,1), (2,3), (3,0),
(3,1), (3,2)
# Всего 4 × 3 = 12 вариантов
```

На ЕГЭ: два применения

1. Перестановка столбцов (№2):

```
# Какая переменная соответствует какому столбцу?
for perm_cols in permutations(range(4)):
    # perm_cols = (2, 0, 1, 3) означает:
    # столбец 0 → переменная 2 (z)
    # столбец 1 → переменная 0 (x)
    # столбец 2 → переменная 1 (y)
    # столбец 3 → переменная 3 (w)
    ...
```

2. Перестановка строк (№2):

```
# В каком порядке идут строки в заданной таблице?
n = len(rows_F1)  # Количество строк с F=1
for perm_rows in permutations(range(n), 3):
    # perm_rows = (1, 0, 2) означает:
```

```
    # первая заданная строка → строка таблицы с индексом 1
    # вторая заданная строка → строка таблицы с индексом 0
    # третья заданная строка → строка таблицы с индексом 2
     ...
```

4. Задание №2: полный разбор

Суть задания

Дано:

1. Логическое выражение F(x, y, z, w).
2. Частично заполненная таблица истинности (3 строки, некоторые значения пропущены — `None`).

Нужно: определить, какой столбец какой переменной соответствует.

Алгоритм решения

1. **Записываем функцию F** на Python.
2. **Строим полную таблицу** из 16 строк через `product([0, 1], repeat=4)`.
3. **Выбираем строки с F=1** в отдельный список `rows_F1`.
4. **Выписываем заданные строки** в виде списка `given` (пустые клетки = `None`).
5. **Перебираем все перестановки столбцов** (24 варианта).
6. **Для каждой перестановки столбцов** перебираем **перестановки индексов строк** (берём 3 строки из `rows_F1` и перебираем их порядок).
7. **Проверяем:** совпадают ли известные значения в `given` со значениями в переставленной таблице.
8. Если совпадают — выводим порядок букв.

Ключевая идея: косвенная адресация

Мы **не переставляем сами данные**. Вместо этого мы переставляем **индексы**:

- `perm_cols[col]` — индекс переменной, которая находится в столбце `col`.
- `perm_rows[i]` — индекс строки в `rows_F1`, которая соответствует i-й строке `given`.

```
full_row = rows_F1[perm_rows[i]]  # Берём строку по индексу
value = full_row[perm_cols[col]]   # Берём значение переменной в столбце
col
```

5. Полный шаблон для задания №2

```
from itertools import product, permutations

# 1. Функция из условия
def F(x, y, z, w):
    return ((x == z) <= ((not y) or w)) == (not ((w <= z) or (x <= y)))
```

```
# 2. Строим полную таблицу истинности
full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

# 3. Строки, где F = 1 (заданная таблица обычно содержит F=1)
rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)  # Количество таких строк

# 4. Заданные строки из условия (None = пропуск)
given = [
    (None, 1, None, 0),
    (0, None, 1, None),
    (0, None, 0, 0),
]

# 5. Имена переменных
vars_names = ['x', 'y', 'z', 'w']

# 6. Внешний цикл: перестановка столбцов (24 варианта)
for perm_cols in permutations(range(4)):
    # 7. Внутренний цикл: перестановка индексов строк
    # Берём 3 строки из rows_F1 и перебираем их порядок
    for perm_rows in permutations(range(n), 3):
        match = True

        # 8. Проверяем все 3 строки
        for i in range(3):
            full_row = rows_F1[perm_rows[i]]   # Строка таблицы
            given_row = given[i]                # Заданная строка

            # Сравниваем известные значения
            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] != given_row[col]:
                        match = False
                        break

            if not match:
                break

        # 9. Если все строки совпали — выводим ответ
        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()  # Завершаем после первого найденного ответа
```

Пошаговый разбор шаблона

Шаг 1. Функция F:

```
def F(x, y, z, w):
    return ((x == z) ⩽ ((not y) or w)) == (not ((w ⩽ z) or (x ⩽ y)))
```

- `⩽` — импликация (следование). `a ⩽ b` истинно всегда, кроме случая `a=1, b=0`.

- `==` — эквивалентность. `a == b` истинно, когда a и b равны.
- `not` — отрицание.

Шаг 2. Полная таблица:

```
pythonfull_table
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))
```

- `product([0, 1], repeat=4)` — 16 наборов (0,0,0,0) до (1,1,1,1).
- Каждый элемент `full_table` — пара `((x, y, z, w), F)`.

Шаг 3. Строки с F=1:

```
rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)
```

- В задании №2 обычно **все заданные строки имеют F=1**. Поэтому мы ищем только среди таких строк.
- Если в задании есть строки с F=0, нужно расширить список.

Шаг 4. Заданные строки:

```
given = [
    (None, 1, None, 0),
    (0, None, 1, None),
    (0, None, 0, 0),
]
```

- `None` означает «любое значение» (пустая клетка в таблице).
- F не включаем в кортеж, потому что F всегда 1 (учтено в шаге 3).

Шаг 5. Имена переменных:

```
vars_names = ['x', 'y', 'z', 'w']
```

Шаг 6–7. Два вложенных цикла:

```
for perm_cols in permutations(range(4)):        # 24 варианта
    for perm_rows in permutations(range(n), 3):  # n!/(n-3)! вариантов
```

- `permutations(range(4))` — все 24 способа сопоставить столбцы переменным.
- `permutations(range(n), 3)` — все способы выбрать 3 строки из n и упорядочить их.

Шаг 8. Проверка:

```
full_row = rows_F1[perm_rows[i]]  # Берём i-ю выбранную строку
if full_row[perm_cols[col]] ≠ given_row[col]:  # Сравниваем значения
```

- `full_row` — кортеж `(x, y, z, w)`.
- `perm_cols[col]` — индекс переменной в `full_row` для столбца `col`.
- Например, `perm_cols = (2, 0, 1, 3)` → `perm_cols[0] = 2` → `full_row[2] = z`.

Шаг 9. Вывод ответа:

```
result = ''.join(vars_names[perm_cols[col]] for col in range(4))
print(result)
```

- `vars_names[perm_cols[col]]` — буква переменной в столбце `col`.
- `''.join( ... )` — склеивает буквы в строку (например, `"wzyx"`).

6. Важно: когда заданных строк много

Проблема

В некоторых вариантах №2 в заданной таблице **много строк** (например, 6 или 8). Тогда перебор размещений `permutations(range(n), k)` может стать слишком большим.

Строк с F=1 (n)	Заданных строк (k)	Вариантов размещений
6	3	6 × 5 × 4 = 120
8	4	8 × 7 × 6 × 5 = 1680
10	5	10 × 9 × 8 × 7 × 6 = 30240
12	6	12 × 11 × 10 × 9 × 8 × 7 = 665280

Решение

1. **Строим полную таблицу на компьютере** (это около 16 строк — мгновенно).
2. **Анализируем заданную таблицу вручную:**
   - Ищем строки, где много известных значений.
   - Сравниваем их со строками полной таблицы.
   - Исключаем невозможные соответствия.
3. **Сужаем перебор** — оставляем только возможные варианты.

Пример ручного анализа:

Дана строка: `(1, 0, None, 1)`. Это означает:

- В столбцах 0, 1, 3 стоят значения 1, 0, 1.
- Ищем в полной таблице строки, где три переменные имеют значения (1, 0, 1) в каком-то порядке.

Часто одна такая строка сразу определяет соответствие столбцов.

> **Совет:** На экзамене сначала постройте полную таблицу (16 строк). Затем посмотрите на заданные строки. Если строк много — используйте код для построения таблицы, а анализ делайте вручную, глядя на выведенные строки.

7. Практика

Задача 1

Функция: `((x ≡ z) → (¬y ∨ w)) ≡ ¬ ((w → z) ∨ (x → y))`

Таблица:

?	?	?	?	F
	1		0	1
0		1		1
0		0	0	1

Код:

```
from itertools import product, permutations

def F(x, y, z, w):
    return ((x == z) <= ((not y) or w)) == (not ((w <= z) or (x <= y)))

full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)

given = [
    (None, 1, None, 0),
    (0, None, 1, None),
    (0, None, 0, 0),
]

vars_names = ['x', 'y', 'z', 'w']

for perm_cols in permutations(range(4)):
    for perm_rows in permutations(range(n), 3):
        match = True
        for i in range(3):
            full_row = rows_F1[perm_rows[i]]
            given_row = given[i]
            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] != given_row[col]:
                        match = False
                        break
            if not match:
                break
        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()
```

Задача 2

Функция: `(¬x ∧ z ∧ ¬y ∧ ¬w) ∨ (¬x ∧ z ∧ y ∧ ¬w) ∨ (¬x ∧ z ∧ y ∧ w)`

Таблица:

?	?	?	?	F
	1	0		1
0	0			1
		1		1

Код (меняем только функцию и таблицу):

```
from itertools import product, permutations

def F(x, y, z, w):
    return (not x and z and not y and not w) or (not x and z and y and
not w) or (not x and z and y and w)

full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)

given = [
    (None, 1, 0, None),
    (0, 0, None, None),
    (None, None, 1, None),
]

vars_names = ['x', 'y', 'z', 'w']

for perm_cols in permutations(range(4)):
    for perm_rows in permutations(range(n), 3):
        match = True
        for i in range(3):
            full_row = rows_F1[perm_rows[i]]
            given_row = given[i]
            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] != given_row[col]:
                        match = False
                        break
            if not match:
                break
        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()
```

Задача 3

Функция: `((x → y) ≡ (y → z)) ∧ (y ∨ w)`

Таблица:

?	?	?	?	F
0		0		1
0	0		0	1
			0	1

Код (меняем только функцию и таблицу):

```
from itertools import product, permutations

def F(x, y, z, w):
    return ((x <= y) == (y <= z)) and (y or w)

full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)

given = [
    (0, None, 0, None),
    (0, 0, None, 0),
    (None, None, None, 0),
]

vars_names = ['x', 'y', 'z', 'w']

for perm_cols in permutations(range(4)):
    for perm_rows in permutations(range(n), 3):
        match = True
        for i in range(3):
            full_row = rows_F1[perm_rows[i]]
            given_row = given[i]
            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] != given_row[col]:
                        match = False
                        break
            if not match:
                break
        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()
```

Домашнее задание

Задание 1

Решите задачу №2 из открытого банка ФИПИ, используя шаблон с двумя перестановками.

Задание 2

Модифицируйте шаблон так, чтобы он работал для функции от **3 переменных** (x, y, z).

Задание 3

Дана таблица с 6 строками (все с F=1). Полную таблицу постройте кодом, а соответствие столбцов найдите **вручную**, анализируя выведенные строки.

Что мы сегодня изучили

Тема	Ключевые моменты
Брутфорс	Полный перебор вариантов, когда их обозримое количество
`itertools.product`	Все комбинации, `repeat` для повторения
`itertools.permutations`	Все перестановки, можно указать длину
Шаблон №2	Два вложенных цикла: столбцы (24) × строки (n!/(n−3)!)
Косвенная адресация	Переставляем индексы, а не данные
Много строк → анализ	Строим таблицу кодом, анализируем вручную

> **Связь с ЕГЭ:** Задание №2 решается этим шаблоном за 3 минуты. Код одинаковый, меняется только функция и таблица.

На следующем занятии (Занятие 9)

Тема: Брутфорс в таблицах истинности (№2)

Чек-лист занятия

- ☐ Понимаю, что такое брутфорс и когда он применим
- ☐ Умею использовать `itertools.product` с `repeat`
- ☐ Умею использовать `itertools.permutations` с выбором длины
- ☐ Знаю шаблон с двумя вложенными циклами для №2
- ☐ Понимаю, что такое косвенная адресация через индексы
- ☐ Выполнено домашнее задание

Занятие 9. Брутфорс в таблицах истинности (задание №2)

План занятия

1. Логические операции в Python
2. Вывод полной таблицы истинности
3. Фильтрация строк по значению F
4. Автоподбор столбцов: финальный шаблон
5. Практика

> **Цель занятия:** Довести до автоматизма решение задания №2. После этого занятия вы должны решать №2 за 2–3 минуты.

1. Логические операции в Python

Таблица операций

Операция	В логике	В Python	Пример
Конъюнкция (И)	`∧`	`and`	`x and y`
Дизъюнкция (ИЛИ)	`∨`	`or`	`x or y`
Отрицание (НЕ)	`¬`	`not`	`not x`
Импликация (→)	`→`	`⩽`	`x ⩽ y`
Эквивалентность (≡)	`≡`	`==`	`x == y`
Исключающее ИЛИ (⊕)	`⊕`	`!=`	`x != y`

Импликация: `x ⩽ y`

Импликация `x → y` ложна **только** когда `x = 1, y = 0`. Во всех остальных случаях — истина.

```
print(0 ⩽ 0)  # True
print(0 ⩽ 1)  # True
print(1 ⩽ 0)  # False  ← единственный случай лжи
print(1 ⩽ 1)  # True
```

Эквивалентность: `x == y`

Эквивалентность `x ≡ y` истинна, когда `x` и `y` равны.

```
print(0 == 0)  # True
print(0 == 1)  # False
print(1 == 0)  # False
print(1 == 1)  # True
```

Исключающее ИЛИ: `x ≠ y`

Исключающее ИЛИ `x ⊕ y` истинно, когда значения **разные**.

```
print(0 ≠ 0)  # False
print(0 ≠ 1)  # True
print(1 ≠ 0)  # True
print(1 ≠ 1)  # False
```

Приоритет операций

От высшего к низшему: `not` → `and` → `or` → `⩽` → `=`

```
# not x and y  →  (not x) and y
# x and y or z →  (x and y) or z
```

> **Всегда ставьте скобки**, если есть сомнения. На экзамене лучше перестраховаться.

2. Вывод полной таблицы истинности

Базовый код

```
from itertools import product

def F(x, y, z, w):
    return (x or y) and not (z and w)

print("x y z w F")
for x, y, z, w in product([0, 1], repeat=4):
    print(x, y, z, w, int(F(x, y, z, w)))
```

Вывод:

```
x y z w F
0 0 0 0 0
0 0 0 1 0
0 0 1 0 0
0 0 1 1 0
0 1 0 0 1
0 1 0 1 1
0 1 1 0 1
0 1 1 1 0
1 0 0 0 1
1 0 0 1 1
1 0 1 0 1
1 0 1 1 0
1 1 0 0 1
1 1 0 1 1
1 1 1 0 1
1 1 1 1 0
```

Сохранение в список для обработки

```
from itertools import product

def F(x, y, z, w):
    return (x or y) and not (z and w)

full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

# full_table[i] = ((x, y, z, w), F)
# Например: full_table[0] = ((0, 0, 0, 0), 0)
```

> **Сохраняем в список**, чтобы потом фильтровать и искать нужные строки.

3. Фильтрация строк по значению F

Строки с F = 1

В задании №2 заданная таблица обычно содержит **только строки с F = 1** (или только с F = 0).

```
# Строки, где F = 1
rows_F1 = [row for row, val in full_table if val == 1]

print(f"Строк с F=1: {len(rows_F1)}")
for row in rows_F1:
    print(f"x={row[0]}, y={row[1]}, z={row[2]}, w={row[3]}")
```

Строки с F = 0 (если нужно)

```
rows_F0 = [row for row, val in full_table if val == 0]
```

Зачем это нужно

Если в заданной таблице все строки имеют F = 1, то мы ищем соответствие **только среди строк с F = 1**. Это сужает перебор.

Если в таблице есть и F = 0, и F = 1, то нужно работать с полной таблицей.

4. Автоподбор столбцов: финальный шаблон

Полный код

```
from itertools import product, permutations

# 1. Функция из условия
def F(x, y, z, w):
    return ((x == z) <= ((not y) or w)) == (not ((w <= z) or (x <= y)))

# 2. Полная таблица
full_table = []
```

```
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

# 3. Строки с F=1
rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)

# 4. Заданная таблица (None = пропуск)
given = [
    (None, 1, None, 0),
    (0, None, 1, None),
    (0, None, 0, 0),
]

vars_names = ['x', 'y', 'z', 'w']

# 5. Двойной перебор
for perm_cols in permutations(range(4)):
    for perm_rows in permutations(range(n), 3):
        match = True

        for i in range(3):
            full_row = rows_F1[perm_rows[i]]
            given_row = given[i]

            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] != given_row[col]:
                        match = False
                        break
            if not match:
                break

        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()
```

Разбор ключевых строк

`perm_cols` — **перестановка столбцов:**

```
perm_cols = (2, 0, 1, 3)
# Столбец 0 → переменная 2 (z)
# Столбец 1 → переменная 0 (x)
# Столбец 2 → переменная 1 (y)
# Столбец 3 → переменная 3 (w)
```

`perm_rows` — **перестановка индексов строк:**

```
perm_rows = (1, 0, 2)
# given[0] → rows_F1[1]
# given[1] → rows_F1[0]
# given[2] → rows_F1[2]
```

Проверка:

```
full_row = rows_F1[perm_rows[i]]  # Берём строку по индексу
if full_row[perm_cols[col]] ≠ given_row[col]:  # Сравниваем значения
```

- `full_row` — кортеж `(x, y, z, w)`.
- `perm_cols[col]` — индекс переменной для столбца `col`.
- `full_row[perm_cols[col]]` — значение этой переменной в выбранной строке.

Формирование ответа:

```
result = ''.join(vars_names[perm_cols[col]] for col in range(4))
# vars_names = ['x', 'y', 'z', 'w']
# perm_cols = (3, 0, 2, 1)
# col=0: vars_names[3] = 'w'
# col=1: vars_names[0] = 'x'
# col=2: vars_names[2] = 'z'
# col=3: vars_names[1] = 'y'
# result = 'wxzy'
```

5. Практика

Задача 1 (F = 0 для всех строк)

Функция: `(x ∧ ¬y) ∨ (y ≡ z) ∨ w`

Таблица:

?	?	?	?	F
		1		0
1	0	0	0	0
1	0		1	0

Анализ: Все строки имеют F=0.

Код:

```
from itertools import product, permutations

def F(x, y, z, w):
    return (x and not y) or (y == z) or w

full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

# Все строки с F=0 (экономия времени)
rows_F0 = [row for row, val in full_table if val == 0]
n = len(rows_F0)

given = [
```

```
    (None, None, 1, None, 0),
    (1, 0, 0, 0, 0),
    (1, 0, None, 1, 0),
]

vars_names = ['x', 'y', 'z', 'w']

for perm_cols in permutations(range(4)):
    for perm_rows in permutations(range(n), 3):
        match = True
        for i in range(3):
            full_row = rows_F0[perm_rows[i]]
            given_row = given[i]
            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] != given_row[col]:
                        match = False
                        break
            if not match:
                break
        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()
```

Задача 2 (F = 1 для всех строк)

Функция: `(x ≡ (y → z)) ∧ (¬w → (x ≡ y))`

Таблица:

?	?	?	?	F
1	0	1	1	1
0	1	1	1	1
0		0		1

Анализ: Все строки имеют F=1. Фильтруем по `rows_F1`.

Код:

```
from itertools import product, permutations

def F(x, y, z, w):
    return (x == (y <= z)) and ((not w) <= (x == y))

full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)

given = [
```

```
    (1, 0, 1, 1, 1),
    (0, 1, 1, 1, 1),
    (0, None, 0, None, 1),
]

vars_names = ['x', 'y', 'z', 'w']

for perm_cols in permutations(range(4)):
    for perm_rows in permutations(range(n), 3):
        match = True
        for i in range(3):
            full_row = rows_F1[perm_rows[i]]
            given_row = given[i]
            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] ≠ given_row[col]:
                        match = False
                        break
            if not match:
                break
        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()
```

Задача 3 (F = 1 для всех строк)

Функция: `(x ∨ y) ∧ ¬(y ≡ z) ∧ ¬w`

Таблица:

?	?	?	?	F
1				1
0	0	1	0	1
	1	0	0	1

Анализ: Все строки имеют F=1. Фильтруем по `rows_F1`.

Код:

```
from itertools import product, permutations

def F(x, y, z, w):
    return (x or y) and not (y == z) and not w

full_table = []
for x, y, z, w in product([0, 1], repeat=4):
    full_table.append(((x, y, z, w), int(F(x, y, z, w))))

rows_F1 = [row for row, val in full_table if val == 1]
n = len(rows_F1)

given = [
```

```
    (1, None, None, None, 1),
    (0, 0, 1, 0, 1),
    (None, 1, 0, 0, 1),
]

vars_names = ['x', 'y', 'z', 'w']

for perm_cols in permutations(range(4)):
    for perm_rows in permutations(range(n), 3):
        match = True
        for i in range(3):
            full_row = rows_F1[perm_rows[i]]
            given_row = given[i]
            for col in range(4):
                if given_row[col] is not None:
                    if full_row[perm_cols[col]] ≠ given_row[col]:
                        match = False
                        break
            if not match:
                break
        if match:
            result = ''.join(vars_names[perm_cols[col]] for col in
range(4))
            print(result)
            exit()
```

Домашнее задание

Задание 1

Решите задачу №2 из банка ФИПИ, используя шаблон.

Задание 2

Модифицируйте шаблон для случая, когда в `given` есть строки **и с F=1, и с F=0**.

Задание 3

Функция: `(x ∧ ¬y) ∨ (y ≡ z) ∨ ¬w`. Таблица (F=0):

?	?	?	?
0		0	0
0	1	0	
	1	0	1

Найдите соответствие столбцов.

Что мы сегодня изучили

Тема	Ключевые моменты
Логические операции	`and`, `or`, `not`, `<=` (→), `==` (≡), `!=` (⊕)
Полная таблица	`product([0, 1], repeat=4)` — 16 строк
Фильтрация	`rows_F1 = [row for row, val in full_table if val == 1]`
Автоподбор	Двойной перебор: столбцы (24) × строки (размещения)
Косвенная адресация	`full_row[perm_cols[col]]` — значение переменной в столбце

> **Связь с ЕГЭ:** Задание №2 — 1 балл. Решается этим шаблоном за 2–3 минуты.

На следующем занятии (Занятие 10)

Тема: Брутфорс в алгоритмах и системах счисления (№5, 14)

- Перебор входного числа N
- Перевод через `bin(N)[2:]` и `int(s, base)`
- Дописывание разрядов, замена цифр
- Перебор неизвестной цифры и основания СС

Чек-лист занятия

- ☐ Знаю все логические операции в Python
- ☐ Умею строить полную таблицу истинности
- ☐ Умею фильтровать строки по F
- ☐ Знаю финальный шаблон для №2
- ☐ Понимаю косвенную адресацию через `perm_cols` и `perm_rows`
- ☐ Выполнено домашнее задание

Занятие 10. Брутфорс в алгоритмах и системах счисления (№5, 14)

План занятия

1. Задание №5: алгоритм в виде функции, перебор N
2. Системы счисления: `bin()`, `oct()`, `hex()`, `int(s, base)`
3. Задание №14: перебор неизвестной цифры и основания

4. Вычисление значений арифметических выражений

5. Практика

> **Цель занятия:** Научиться решать задания №5 и №14 брутфорсом. Это два стабильных первичных балла.

1. Задание №5: алгоритм в виде функции, перебор N

Общий подход

1. Описываем алгоритм из условия в виде функции `algo(N)`, которая возвращает `R`.
2. Перебираем `N` в разумном диапазоне.
3. Ищем `N`, при котором `R` удовлетворяет условию.

Шаблон для №5

```
def algo(N):
    # Шаг 1: перевод в нужную СС
    s = ''
    temp = N
    while temp > 0:
        s = str(temp % BASE) + s
        temp //= BASE

    # Шаг 2–3: обработка по условию
    if условие:
        s = s + ...
    else:
        s = s + ...

    # Шаг 4: перевод обратно в десятичную
    return int(s, BASE)

# Перебор
for N in range(1, 1000):
    R = algo(N)
    if R > 150:
        print(N)
        break
```

Перевод в СС вручную (для нестандартных оснований)

```
def to_base(n, base):
    """Перевод числа n в систему счисления с основанием base."""
    if n == 0:
        return '0'
    digits = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    result = ''
    while n > 0:
        result = digits[n % base] + result
        n //= base
    return result
```

Важно: `int(s, base)` работает только для base ≤ 36

Для больших оснований используйте самописную функцию `to_base()` и `int()` для обратного перевода не сработает — придётся писать `from_base()`.

2. Системы счисления: `bin()`, `oct()`, `hex()`, `int(s, base)`

Встроенные функции

Функция	Назначение	Пример	Результат
`bin(N)`	В двоичную	`bin(42)`	`'0b101010'`
`oct(N)`	В восьмеричную	`oct(42)`	`'0o52'`
`hex(N)`	В шестнадцатеричную	`hex(42)`	`'0×2a'`
`int(s, base)`	Из любой в десятичную	`int('101', 2)`	`5`

Убираем префиксы

```
binary  = bin(N)[2:]   # '101010'
octal   = oct(N)[2:]   # '52'
hex_str = hex(N)[2:]   # '2a'
```

Дописывание разрядов (№5)

```
s = bin(N)[2:]          # Двоичная запись
if N % 2 == 0:
    s += '0'            # Дописать 0
else:
    s += '1'            # Дописать 1
R = int(s, 2)           # Обратно в десятичную
```

Замена цифр в строке (№5, 14)

```
s = s.replace('0', 'x')  # Временно меняем 0 на x
s = s.replace('2', '0')  # Меняем 2 на 0
s = s.replace('x', '2')  # Меняем x (бывшие 0) на 2
# Или через генератор:
s = ''.join('2' if c == '0' else ('0' if c == '2' else c) for c in s)
```

3. Задание №14: перебор неизвестной цифры и основания

Перебор цифры `x`

```
# Выражение: 1x2_3 + 3×1_4 = 25
for x in range(0, 10):  # x — цифра, от 0 до 9
    a = int(f"1{x}2", 3)
    b = int(f"3{x}1", 4)
    if a + b == 25:
        print(f"x = {x}")
```

Перебор основания

```
# Выражение: 123_base = 66_10
for base in range(4, 37):  # Основание больше максимальной цифры (3)
    try:
        value = int('123', base)
        if value == 66:
            print(f"Основание: {base}")
    except ValueError:
        pass  # Цифра не подходит для этого основания
```

Ограничение: `int(s, base)` требует base ≤ 36 и цифры 0–9, A–Z

Для оснований > 36 используйте самописную функцию:

4. Вычисление значений арифметических выражений

Степени и большие числа

Python работает с большими целыми числами автоматически:

```
x = 2**1000  # Огромное число, Python справится
```

Подсчёт цифр в записи

```
n = 8**5 + 4**6 + 2**12 - 16
binary = bin(n)[2:]
print(f"Единиц в двоичной записи: {binary.count('1')}")
```

Подсчёт нулей в СС с основанием base

```
def count_zeros(n, base):
    """Считает количество нулей в записи числа n в СС с основанием
base."""
    count = 0
    while n > 0:
        if n % base == 0:
            count += 1
        n //= base
    return count
```

5. Практика

Задача 1 (№5, троичная система)

Условие: Алгоритм для N:

1. Троичная запись N.
2. Если N кратно 3 → дописать три последние цифры.
3. Если N не кратно 3 → остаток × 3, перевести в троичную, дописать.

Найти: минимальное N, при котором R > 150.

```
def algo(N):
    # 1. Троичная запись
    ternary = ''
    temp = N
    while temp > 0:
        ternary = str(temp % 3) + ternary
        temp //= 3

    # 2–3. Обработка
    if N % 3 == 0:
        ternary += ternary[-3:]  # Три последние цифры
    else:
        remainder = N % 3
        add = remainder * 3
        # Перевод add в троичную
        add_ternary = ''
        while add > 0:
            add_ternary = str(add % 3) + add_ternary
            add //= 3
        ternary += add_ternary

    # 4. В десятичную
    return int(ternary, 3)

for N in range(1, 200):
    R = algo(N)
    if R > 150:
        print(f"N = {N}, R = {R}")
        break
```

Задача 2 (№5, тройки цифр)

Условие: Алгоритм для N ≥ 100:

1. Все тройки соседних цифр → трёхзначные числа.
2. Наибольшее − наименьшее = R.

Найти: наименьшее N, при котором R = 415.

```
def algo(N):
    s = str(N)
    triples = []
    for i in range(len(s) - 2):
        triples.append(int(s[i:i+3]))
    return max(triples) - min(triples)

N = 100
while True:
    if algo(N) == 415:
        print(f"N = {N}")
        break
    N += 1
```

Задача 3 (№5, троичная, сумма цифр)

Условие: Алгоритм для N:

1. Троичная запись. 2а. Если N кратно 3 → дописать две последние цифры. 2б. Если N не кратно 3 → сумма цифр × 3, в троичную, дописать.
2. Результат R в десятичной.

Найти: R, ближайшее к 910.

```
def algo(N):
    # Троичная запись
    ternary = ''
    temp = N
    while temp > 0:
        ternary = str(temp % 3) + ternary
        temp //= 3

    if N % 3 == 0:
        ternary += ternary[-2:]
    else:
        digit_sum = sum(int(d) for d in ternary)
        add = digit_sum * 3
        add_ternary = ''
        while add > 0:
            add_ternary = str(add % 3) + add_ternary
            add //= 3
        ternary += add_ternary

    return int(ternary, 3)

results = []
for N in range(1, 500):
    results.append(algo(N))

target = 910
closest = min(results, key=lambda r: abs(r - target))
print(f"Ближайшее к {target}: {closest}")
```

Задача 4 (№14, максимум нулей)

Условие: `27**298 + 27**269 - x`, где $0 < x \le 7290$, записали в 27-ричной СС. Найти максимальное количество нулей.

```
def count_zeros_in_base(n, base):
    count = 0
    while n > 0:
        if n % base == 0:
            count += 1
        n //= base
    return count

max_zeros = 0
for x in range(1, 7291):  # x от 0 до 7290
```

```
    value = 27**298 + 27**269 - x
    zeros = count_zeros_in_base(value, 27)
    if zeros > max_zeros:
        max_zeros = zeros

print(f"Максимум нулей: {max_zeros}")
```

Задача 5 (№14, единицы в двоичной)

Условие: Сколько единиц в двоичной записи `8**5 + 4**6 + 2**12 - 16`?

```
value = 8**5 + 4**6 + 2**12 - 16
binary = bin(value)[2:]
print(f"Единиц: {binary.count('1')}")
```

Задача 6 (№14, две переменные и кратность)

Условие: $yx320_7 + 1x3y3_9$. Найти x и y, при которых сумма минимальна и кратна 181. Вывести частное от деления на 181.

```
min_value = float('inf')
answer = None

for x in range(0, 7):  # x — цифра в 7-ричной и 9-ричной (0..6)
    for y in range(0, 7):  # y — цифра в 7-ричной (0..6)
        try:
            a = int(f"{y}{x}320", 7)
            b = int(f"1{x}3{y}3", 9)
            total = a + b
            if total % 181 == 0 and total < min_value:
                min_value = total
                answer = total // 181
        except ValueError:
            pass

print(f"Частное: {answer}")
```

> **Важно:** В задачах с неизвестными цифрами перебираем x и y в допустимых диапазонах. Для СС с основанием `b` цифры от 0 до `b-1`.

Домашнее задание

Задание 1 (№5)

Алгоритм: двоичная запись N. Если N чётно — дописать '01', иначе '10'. Найти минимальное N, при котором R > 100.

Задание 2 (№5)

Алгоритм: десятичная запись N. Удалить все нечётные цифры. Результат R. Найти N, при котором R = 248.

Задание 3 (№14)

`2×1_3 + 1×2_4 = 30`. Найти x.

Что мы сегодня изучили

Тема	Ключевые моменты
№5: алгоритм в функцию	`def algo(N): return R`, перебор N
Перевод в СС	`bin()`, `oct()`, `hex()`, `int(s, base)`, ручной перевод
Дописывание разрядов	`s += '0'`, `s[-3:]`, замена цифр
№14: перебор x	`int(f"...", base)` с подстановкой x
№14: перебор основания	`for base in range(...)`
Большие числа	Python работает с ними автоматически

> **Связь с ЕГЭ:** Задания №5 и №14 — два стабильных балла. Брутфорс решает их за 3–5 минут каждое.

На следующем занятии (Занятие 11)

Тема: Брутфорс в комбинаторике (задание №8, простые)

- `itertools.product(alphabet, repeat=L)`
- Фильтрация по `.count()`, проверка подстрок
- Оценка объёма перебора

Чек-лист занятия

- ☐ Умею оформлять алгоритм №5 в функцию и перебирать N
- ☐ Знаю `bin()`, `oct()`, `hex()`, `int(s, base)`
- ☐ Умею дописывать разряды и заменять цифры в строке
- ☐ Умею перебирать неизвестную цифру и основание в №14
- ☐ Знаю, как считать нули/единицы в записи числа
- ☐ Выполнено домашнее задание

Занятие 11. Брутфорс в комбинаторике (задание №8, простые)

План занятия

1. `itertools.product` — генерация всех слов
2. Фильтрация по условиям: `.count()`, `.find()`, проверка подстрок
3. Нумерация слов в алфавитном порядке
4. Оценка объёма перебора: когда `product` работает
5. Практика

> **Цель занятия:** Научиться решать простое задание №8 брутфорсом за 3–5 минут.

1. `itertools.product` — генерация всех слов

Базовый синтаксис

```
from itertools import product

# Все слова длины 3 из букв А, Б
for word in product('АБ', repeat=3):
    print(''.join(word))
```

Вывод:

```
ААА
ААБ
АБА
АББ
БАА
БАБ
ББА
БББ
```

Важные особенности

- `product` генерирует слова в **алфавитном порядке** (в порядке переданной строки).
- `repeat=L` — длина слова.
- Результат — кортеж символов. Чтобы получить строку: `''.join(word)`.

Оценка количества слов

Длина (L)	Алфавит (N)	Количество слов	Время перебора
4	5	$5^4 = 625$	Мгновенно
5	4	$4^5 = 1024$	Мгновенно
6	6	$6^6 = 46656$	Мгновенно
7	8	$8^7 \approx 2$ млн	~0.1 секунды
8	8	$8^8 \approx 16$ млн	~1 секунда
10	10	$10^{10} = 10$ млрд	Слишком долго

> **Правило:** Если $N^L \leq 10^7$ — брутфорс работает. Если больше — нужно ДП (занятие 25).

2. Фильтрация по условиям

`.count()` — подсчёт символов

```
word = 'АБАБА'
print(word.count('А'))  # 3
print(word.count('Б'))  # 2
```

Проверка подстрок

```
# Конкретная подстрока
if 'АА' not in word:
    ...

# Несколько подстрок
if 'АА' not in word and 'ББ' not in word:
    ...
```

Проверка соседних символов

```
# Проверка, что никакая нечётная цифра не стоит рядом с цифрой 6
def check(word):
    for i in range(len(word) - 1):
        if word[i] == '6' and word[i+1] in '1357':
            return False
        if word[i] in '1357' and word[i+1] == '6':
            return False
    return True
```

Типовые условия в №8

Условие	Код
Ровно k букв 'A'	`word.count('A') == k`
Не более k букв 'A'	`word.count('A') <= k`
Все буквы различны	`len(set(word)) == len(word)`
Нет подстроки 'AA'	`'AA' not in word`
Согласная не рядом с гласной	Проверка пар соседей

3. Нумерация слов в алфавитном порядке

Задача: найти слово по номеру

`product` генерирует слова в алфавитном порядке. Поэтому:

- Слово №1 — первое (`index = 0`).
- Слово №170 — `index = 169` .

```
from itertools import product

alphabet = 'АОУ'
L = 5
target = 170  # Номер слова (нумерация с 1)

for i, word in enumerate(product(alphabet, repeat=L)):
    if i + 1 == target:  # i+1 — номер с 1
        print(''.join(word))
        break
```

Быстрый способ (без перебора всех)

Можно перевести номер в систему счисления с основанием = длина алфавита:

```
def word_by_number(alphabet, L, n):
    """Возвращает слово № n (нумерация с 1) длины L из alphabet."""
    n -= 1  # Переходим к нумерации с 0
    result = []
    base = len(alphabet)
    for _ in range(L):
        result.append(alphabet[n % base])
        n //= base
    return ''.join(reversed(result))

print(word_by_number('АОУ', 5, 170))
```

> Этот метод работает мгновенно для любой длины. Полезен, если брутфорс слишком долгий.

4. Оценка объёма перебора

Когда `product` справляется

```
# Быстро: 6⁵ = 7776 слов
product('012345', repeat=5)  # ~0.01 сек

# Быстро: 8⁶ = 262144 слов
product('01234567', repeat=6)  # ~0.05 сек

# Медленно: 10⁷ = 10 млн слов
product('0123456789', repeat=7)  # ~1–2 сек

# Невозможно: 15⁸ ≈ 2.5 млрд слов
product('АБВГДЕЁЖЗИЙКЛМНО', repeat=8)  #
```

Когда `product` НЕ справляется

- Длина слова > 8 и алфавит > 10.
- Тогда нужно **динамическое программирование** (занятие 25).

Что делать, если объём большой

1. Сократить алфавит (например, гласные/согласные вместо конкретных букв).
2. Использовать ДП по длине слова.
3. Использовать формулу нумерации (для задач «слово по номеру»).

5. Практика

Задача 1 (слово по номеру)

Условие: Все 5-буквенные слова из А, О, У в алфавитном порядке. Какое слово на 170-м месте?

Код (брутфорс):

```
from itertools import product

alphabet = 'АОУ'
target = 170

for i, word in enumerate(product(alphabet, repeat=5)):
    if i + 1 == target:
        print(''.join(word))
        break
```

Код (быстрый):

```
def word_by_number(alphabet, L, n):
    n -= 1
    result = []
    base = len(alphabet)
```

```
    for _ in range(L):
        result.append(alphabet[n % base])
        n //= base
    return ''.join(reversed(result))

print(word_by_number('АОУ', 5, 170))
```

Задача 2 (одна цифра 6, нечётные не рядом)

Условие: Пятизначные числа в 8-ричной СС. Ровно одна цифра 6, нечётные не рядом с 6.

```
from itertools import product

digits = '01234567'
odd_digits = '1357'
count = 0

for word in product(digits, repeat=5):
    # Первая цифра не 0
    if word[0] == '0':
        continue

    # Ровно одна цифра 6
    if word.count('6') != 1:
        continue

    # Нечётная не рядом с 6
    ok = True
    for i in range(4):
        if word[i] == '6' and word[i+1] in odd_digits:
            ok = False
            break
        if word[i] in odd_digits and word[i+1] == '6':
            ok = False
            break

    if ok:
        count += 1

print(count)
```

Задача 3 (все цифры различны, чёт/нечет чередуются)

Условие: Четырёхзначные десятичные числа. Все цифры различны, чётные и нечётные не стоят рядом.

```
from itertools import product

digits = '0123456789'
even = '02468'
odd = '13579'
count = 0

for word in product(digits, repeat=4):
    if word[0] == '0':
        continue
```

```
    # Все цифры различны
    if len(set(word)) != 4:
        continue

    # Чётные и нечётные чередуются
    ok = True
    for i in range(3):
        # Если соседи оба чётные или оба нечётные — нарушение
        if (word[i] in even and word[i+1] in even) or \
           (word[i] in odd and word[i+1] in odd):
            ok = False
            break

    if ok:
        count += 1

print(count)
```

Задача 4 (ровно две тройки, нечётные не рядом с 2)

Условие: Пятизначные числа в 9-ричной СС. Ровно две цифры 3, нечётные не рядом с 2.

```
from itertools import product

digits = '012345678'
odd = '1357'
count = 0

for word in product(digits, repeat=5):
    if word[0] == '0':
        continue

    # Ровно две цифры 3
    if word.count('3') != 2:
        continue

    # Нечётные не рядом с 2
    ok = True
    for i in range(4):
        if word[i] == '2' and word[i+1] in odd:
            ok = False
            break
        if word[i] in odd and word[i+1] == '2':
            ok = False
            break

    if ok:
        count += 1

print(count)
```

Задача 5 (X только на первом месте или нигде)

Условие: 4-буквенные слова из A, B, C, D, X. X может быть только на первом месте или отсутствовать.

```python
from itertools import product

alphabet = 'ABCDX'
count = 0

for word in product(alphabet, repeat=4):
    # X не может быть на позициях 2, 3, 4 (индексы 1, 2, 3)
    if 'X' in word[1:]:
        continue
    count += 1

print(count)
```

Аналитическое решение (для проверки):

- X на первом месте: остальные 3 буквы из A, B, C, D → $4^3 = 64$.
- X нет вообще: все 4 буквы из A, B, C, D → $4^4 = 256$.
- Всего: $64 + 256 = 320$.

Задача 6 (слово по номеру, 239-е место)

Условие: 5-буквенные слова из Б, К, Ф, Ц. Какое слово на 239-м месте?

```python
from itertools import product

alphabet = 'БКФЦ'
target = 239

for i, word in enumerate(product(alphabet, repeat=5)):
    if i + 1 == target:
        print(''.join(word))
        break
```

Быстрый способ:

```python
def word_by_number(alphabet, L, n):
    n -= 1
    result = []
    base = len(alphabet)
    for _ in range(L):
        result.append(alphabet[n % base])
        n //= base
    return ''.join(reversed(result))

print(word_by_number('БКФЦ', 5, 239))
```

Домашнее задание

Задание 1

Все 4-буквенные слова из А, Б, В в алфавитном порядке. Какое слово на 50-м месте?

Задание 2

Сколько 6-буквенных слов из букв А, Б, В, Г, Д, в которых ровно две буквы А и нет подстроки «ББ»?

Задание 3

Сколько пятизначных чисел в 7-ричной СС, где все цифры различны и нет цифры 0 на первом месте?

Что мы сегодня изучили

Тема	Ключевые моменты
`product`	Генерация всех слов длины L из алфавита
Фильтрация	`.count()`, `in`, проверка пар соседей
Нумерация	`enumerate(product(...))` или формула СС
Оценка объёма	До 10^7 — брутфорс, больше — ДП

> **Связь с ЕГЭ:** Задание №8 (простое) — 1 балл. Решается брутфорсом за 3–5 минут.

На следующем занятии (Занятие 12)

Тема: Брутфорс в логике (задание №15, А — число)

- Функция `F(x, A)`
- `all()` для проверки всех x
- Поиск минимального/максимального А

Чек-лист занятия

- ☐ Умею генерировать слова через `product`
- ☐ Умею фильтровать по `.count()`, подстрокам, соседям
- ☐ Знаю, как найти слово по номеру
- ☐ Понимаю, когда брутфорс применим, а когда нет
- ☐ Выполнено домашнее задание

Занятие 12. Брутфорс в логике (задание №15, А — число)

План занятия

1. Когда А — число: брутфорс выгоден

2. Функция `F(x, A)` и `all()` для проверки всех x

3. Поиск минимального/максимального A

4. Задачи с двумя переменными

5. Делимость и поразрядная конъюнкция

6. Практика

> **Цель занятия:** Научиться решать задание №15 брутфорсом, когда A — число. Это 1 стабильный балл.

1. Когда A — число: брутфорс выгоден

Два типа задания №15

Тип	A	Метод
A — число	Одно целое число	Брутфорс на Python
A — отрезок	Пара чисел [start, end]	Вручную (диаграммы Эйлера)

Как определить тип

- «Для какого **наибольшего/наименьшего целого числа A**...» → брутфорс.
- «Найдите **длину отрезка A**...» → вручную.

> **Правило:** Если в вопросе одно число — решаем кодом. Если отрезок — решаем руками.

2. Функция `F(x, A)` и `all()` для проверки всех x

Шаблон для A — числа

```
def F(x, y, A):
    return (x * y < 140) or (y > A) or (x > A)

for A in range(1, 1000):
    if all(F(x, y, A) for x in range(1, 100) for y in range(1, 100)):
        print(A)
```

`all()` — проверка, что условие верно для всех элементов

```
# all(выражение for переменная in коллекция)
# Возвращает True, если ВСЕ значения True

print(all(x > 0 for x in [1, 2, 3]))  # True
print(all(x > 0 for x in [1, -2, 3]))  # False
```

`any()` — проверка, что условие верно хотя бы для одного

```
print(any(x < 0 for x in [1, 2, 3]))  # False
print(any(x < 0 for x in [1, -2, 3]))  # True
```

Важно: диапазон перебора x и y

- Для неотрицательных целых: `range(0, 100)` или `range(0, 200)`.
- Для натуральных: `range(1, 100)`.
- Диапазон должен быть достаточно большим, чтобы покрыть все возможные значения. Обычно 100–500 хватает.
- Если выражение содержит множители (например, `x * A`), диапазон для x можно расширить.

3. Поиск минимального/максимального A

Поиск наибольшего A

Перебираем A от большего к меньшему, останавливаемся на первом подходящем:

```
for A in range(1000, 0, -1):  # От 1000 вниз до 1
    if all(F(x, A) for x in range(1, 100)):
        print(A)
        break
```

Поиск наименьшего A

Перебираем A от меньшего к большему:

```
for A in range(1, 1000):  # От 1 вверх
    if all(F(x, A) for x in range(1, 100)):
        print(A)
        break
```

Сбор всех подходящих A

Если нужно найти все подходящие A:

```
result = []
for A in range(1, 1000):
    if all(F(x, A) for x in range(1, 100)):
        result.append(A)
print(max(result))  # Наибольшее
print(min(result))  # Наименьшее
```

4. Задачи с двумя переменными

Шаблон для двух переменных

```
def F(x, y, A):
    return (x * y < 140) or (y > A) or (x > A)
```

```
for A in range(1, 500):
    if all(F(x, y, A) for x in range(1, 100) for y in range(1, 100)):
        print(A)
```

Осторожно: двойной цикл в `all()`

```
# Это создаёт 100 × 100 = 10 000 проверок для каждого A
# Для A от 1 до 500: 5 миллионов проверок — нормально

# Если диапазон больше — может быть медленно
```

5. Делимость и поразрядная конъюнкция

ДЕЛ(n, m) — делимость

```
def DEL(n, m):
    return n % m == 0
```

Поразрядная конъюнкция `&`

```
# 14 & 5 = 4
# 14 = 1110₂, 5 = 0101₂ → 0100₂ = 4

print(14 & 5)  # 4

# Условие: x & A > 0 — хотя бы один общий бит
# Условие: x & A == 0 — нет общих битов
```

Импликация в задачах с делимостью

```
# «ДЕЛ(x, 36) → (¬ДЕЛ(x, A) → ¬ДЕЛ(x, 45))»
# В Python:
# (x % 36 == 0) ⩽ ((x % A ≠ 0) ⩽ (x % 45 ≠ 0))
```

6. Практика

Задача 1 (наибольшее A, две переменные)

Условие: `(x * y < 140) v (y > A) v (x > A)` тождественно истинно при любых целых неотрицательных x и y.

Найти: наибольшее A.

```
def F(x, y, A):
    return (x * y < 140) or (y > A) or (x > A)

for A in range(200, 0, -1):  # От большего к меньшему
    if all(F(x, y, A) for x in range(0, 100) for y in range(0, 100)):
        print(f"Наибольшее A = {A}")
        break
```

Разбор:

- `x * y < 140` — основное условие. Если оно ложно, то `x*y ≥ 140`.
- Тогда должно выполняться `(y > A) or (x > A)`.
- Чтобы это работало для любых x, y с `x*y ≥ 140`, A должно быть меньше максимума из (x, y) для всех таких пар.
- Код находит ответ перебором.

Задача 2 (наибольшее A, делимость)

Условие: `ДЕЛ(120, A) ∧ (ДЕЛ(x, 36) → (¬ДЕЛ(x, A) → ¬ДЕЛ(x, 45)))` тождественно истинна.

Найти: наибольшее A.

```
def DEL(n, m):
    return n % m == 0

def F(x, A):
    return DEL(120, A) and ((DEL(x, 36)) ⩽ ((not DEL(x, A)) ⩽ (not
DEL(x, 45))))

for A in range(200, 0, -1):
    if all(F(x, A) for x in range(1, 500)):
        print(f"Наибольшее A = {A}")
        break
```

Разбор:

- `ДЕЛ(120, A)` — A должно быть делителем 120.
- Импликация: `ДЕЛ(x, 36) → (¬ДЕЛ(x, A) → ¬ДЕЛ(x, 45))`.
- Если x делится на 36, и x НЕ делится на A, то x НЕ должно делиться на 45.
- Перебираем x от 1 до 500 (кратные 36, 45, A).

Задача 3 (наименьшее A, поразрядная конъюнкция)

Условие: `((x & 45 > 0) ∨ (x & 89 > 0)) → (x & A > 0)` тождественно истинна.

Найти: наименьшее A.

```
def F(x, A):
    return ((x & 45 > 0) or (x & 89 > 0)) ⩽ (x & A > 0)

for A in range(0, 1000):
    if all(F(x, A) for x in range(0, 500)):
        print(f"Наименьшее A = {A}")
        break
```

Разбор:

- `x & 45 > 0` — у x есть общие биты с 45 (101101_2).
- `x & 89 > 0` — у x есть общие биты с 89 (1011001_2).

- Если у x есть общие биты с 45 ИЛИ с 89, то должны быть общие биты с A.
- Значит, A должно содержать все биты, которые есть в 45 ИЛИ 89.
- `A = 45 | 89 = 125` — аналитический ответ. Код подтверждает.

Задача 4 (наименьшее A, линейное неравенство)

Условие: `(y > 10) v (x * A > y + x)` тождественно истинна при любых натуральных x и y.

Найти: наименьшее A.

```
def F(x, y, A):
    return (y > 10) or (x * A > y + x)

for A in range(1, 200):
    if all(F(x, y, A) for x in range(1, 100) for y in range(1, 100)):
        print(f"Наименьшее A = {A}")
        break
```

Разбор:

- Если `y ≤ 10`, то должно выполняться `x * A > y + x`.
- `x * A > y + x` → `x * (A - 1) > y`.
- При `y = 10` и малых x: `A - 1 > 10 / x`. Для x = 1: `A > 11`.
- Код находит точное значение.

Сводка: диапазоны перебора

Тип задачи	Диапазон A	Диапазон x, y
Неравенства	1..500	0..100
Делимость	1..200	1..500
Поразрядная конъюнкция	0..255	0..255
Линейные выражения	1..200	1..100

> Если не уверены в диапазоне — расширьте его. Python справится с миллионом проверок за доли секунды.

Домашнее задание

Задание 1

Для какого наименьшего целого неотрицательного A выражение `(x + 2*y < A) v (y > 20) v (x > 30)` тождественно истинно при любых целых неотрицательных x и y?

Задание 2

Для какого наименьшего A формула `ДЕЛ(90, A) ∧ (¬ДЕЛ(x, A) → (ДЕЛ(x, 15) → ¬ДЕЛ(x, 20)))` тождественно истинна?

Задание 3

Для какого наименьшего A формула `(x & 25 ≠ 0) → ((x & 17 == 0) → (x & A ≠ 0))` тождественно истинна?

Что мы сегодня изучили

Тема	Ключевые моменты
A — число vs отрезок	Число → брутфорс, отрезок → вручную
`all()`	Проверка истинности для всех x
Поиск max/min A	`range(1000, 0, -1)` для max, `range(1, 1000)` для min
Две переменные	`all( ... for x in ... for y in ... )`
Делимость	`n % m == 0`
Поразрядная конъюнкция	`&`, условие `x & A > 0`

> **Связь с ЕГЭ:** Задание №15 (A — число) — 1 балл. Решается брутфорсом за 3–5 минут.

На следующем занятии (Занятие 13)

Тема: Брутфорс с оптимизацией (задание №25)

- Поиск делителей до √n
- Маски: `str(num)[-2:] == '12'`
- Условия на количество и сумму делителей

Чек-лист занятия

- ☐ Умею определять, когда A — число (брутфорс), а когда отрезок (вручную)
- ☐ Знаю шаблон с `all()` для проверки всех x
- ☐ Умею искать наибольшее и наименьшее A перебором
- ☐ Умею работать с двумя переменными в `all()`
- ☐ Знаю, как записать делимость и поразрядную конъюнкцию
- ☐ Выполнено домашнее задание

Занятие 13. Брутфорс с оптимизацией (задание №25)

План занятия

1. Поиск делителей: перебор до $\sqrt{n}$
2. Проверка на простоту
3. Маски: `?` и `*`
4. Условия на количество и сумму делителей
5. Практика

> **Цель занятия:** Научиться решать задание №25 — поиск чисел с заданными свойствами. Это 1 первичный балл.

1. Поиск делителей: перебор до $\sqrt{n}$

Почему до $\sqrt{n}$

Делители числа n образуют пары: если d — делитель, то n/d — тоже делитель. Меньший в паре не превышает $\sqrt{n}$. Поэтому достаточно перебирать до $\sqrt{n}$.

```
import math

def get_divisors(n):
    """Возвращает список всех делителей числа n."""
    divisors = []
    for d in range(1, int(n**0.5) + 1):  # или math.isqrt(n) + 1
        if n % d == 0:
            divisors.append(d)
            if d != n // d:  # Не добавляем дважды для точного квадрата
                divisors.append(n // d)
    return sorted(divisors)
```

Сравнение скорости

n	Перебор до n	Перебор до $\sqrt{n}$
10^6	1 млн итераций	1000 итераций
10^9	1 млрд (долго)	31622 итерации
10^{12}	Невозможно	1 млн итераций

`math.isqrt(n)` vs `int(n**0.5)`

```
import math

# math.isqrt(n) — целочисленный квадратный корень (точный)
limit = math.isqrt(n)  # Рекомендуется для ЕГЭ
```

```
# int(n**0.5) — может дать погрешность для больших n
limit = int(n**0.5)    # Приемлемо для n до 10^9
```

> **Используйте `math.isqrt(n)`** — это точно и не требует `int()`.

2. Проверка на простоту

Функция `is_prime`

```
import math

def is_prime(n):
    """Возвращает True, если n — простое число."""
    if n < 2:
        return False
    for d in range(2, math.isqrt(n) + 1):
        if n % d == 0:
            return False
    return True
```

Максимальный простой делитель (не равный самому числу)

```
def max_prime_divisor(n):
    """Возвращает максимальный простой делитель числа n (не равный n)."""
    max_prime = 0
    for d in range(2, math.isqrt(n) + 1):
        if n % d == 0:
            if is_prime(d):
                max_prime = max(max_prime, d)
            other = n // d
            if other != d and is_prime(other):
                max_prime = max(max_prime, other)
    if max_prime == 0 and is_prime(n):
        return 0  # Простое число — нет делителей кроме себя
    return max_prime
```

> **Оптимизация:** Для больших чисел лучше сначала найти все делители, а потом проверить их на простоту.

3. Маски: `?` и `*`

- Что такое маска

  - `?` — ровно одна произвольная цифра.
  - `*` — любая последовательность цифр (в том числе пустая).

Подход к перебору маски

Маску можно перебирать **посимвольно**:

```
# Маска: 5?34?71*2
# Фиксированные части: '5', '34', '71', '2'
# Переменные: '?' — одна цифра, '*' — любая строка
```

Перебор через рекурсию или `product`

Для `?`:

```
from itertools import product

# Два знака '?' — две цифры от 0 до 9
for d1, d2 in product('0123456789', repeat=2):
    num_str = f"5{d1}34{d2}71{...}2"
    ...
```

Для `*`:

`*` — самая сложная часть. Это может быть 0, 1, 2, ... цифр. Перебираем длину и все возможные последовательности:

```
# '*' может быть пустой строкой или строкой из цифр 0..9 любой длины
# Для ограничения сверху (число ≤ 10^10) перебираем длину * от 0 до
оставшихся разрядов
```

4. Условия на количество и сумму делителей

Количество делителей

```
def count_divisors(n):
    count = 0
    for d in range(1, math.isqrt(n) + 1):
        if n % d == 0:
            count += 1 if d * d == n else 2
    return count
```

Сумма делителей

```
def sum_divisors(n):
    total = 0
    for d in range(1, math.isqrt(n) + 1):
        if n % d == 0:
            total += d
            if d != n // d:
                total += n // d
    return total
```

Ровно 3 делителя → квадрат простого числа

Число имеет ровно 3 делителя тогда и только тогда, когда оно является **квадратом простого числа**: $n = p^2$. Делители: $1, p, p^2$.

```
# Проверка: ровно 3 делителя
def has_three_divisors(n):
```

```
    root = math.isqrt(n)
    return root * root == n and is_prime(root)
```

5. Практика

Задача 1 (максимальный простой делитель)

Условие: M — максимальный простой делитель числа (не равный самому числу), или 0. Найти первые 5 чисел > 1 825 000, для которых M ≤ 25 000 и M оканчивается на 3.

```
import math

def is_prime(n):
    if n < 2:
        return False
    for d in range(2, math.isqrt(n) + 1):
        if n % d == 0:
            return False
    return True

def max_prime_divisor(n):
    max_p = 0
    for d in range(2, math.isqrt(n) + 1):
        if n % d == 0:
            if is_prime(d):
                max_p = max(max_p, d)
            other = n // d
            if other != d and is_prime(other):
                max_p = max(max_p, other)
    if max_p == 0 and is_prime(n):
        return 0
    return max_p

found = 0
n = 1825001
while found < 5:
    M = max_prime_divisor(n)
    if M <= 25000 and M % 10 == 3:
        print(n)
        found += 1
    n += 1
```

Задача 2 (маска с `?` и `*`, делимость на 2026)

Условие: Маска `5?34?71*2`, число ≤ 10^{10}, делится на 2026.

```
from itertools import product

results = []
odd_digits = '13579'  # Нечётные цифры для '?'
all_digits = '0123456789'

# Два знака '?' — две нечётные цифры
for d1 in odd_digits:
```

```
    for d2 in odd_digits:
        prefix = f"5{d1}34{d2}71"
        # '*' — любая последовательность (включая пустую)
        # Число ≤ 10^10, prefix = 7 цифр + '2' = 8 цифр
        # максимум ещё 2 цифры на '*'
        for star_len in range(0, 3):
            if star_len == 0:
                num = int(prefix + '2')
                if num <= 10**10 and num % 2026 == 0:
                    results.append(num)
            else:
                for star_digits in product(all_digits, repeat=star_len):
                    star = ''.join(star_digits)
                    num = int(prefix + star + '2')
                    if num <= 10**10 and num % 2026 == 0:
                        results.append(num)

for r in sorted(set(results)):
    print(r)
```

Задача 3 (ровно 3 простых множителя, цифры 2 или 3)

Условие: Найти первые 5 чисел > 5 000 000, представимых как произведение ровно трёх простых множителей (необязательно различных), каждый из которых содержит в записи хотя бы одну цифру 2 или 3.

```
import math

def is_prime(n):
    if n < 2:
        return False
    for d in range(2, math.isqrt(n) + 1):
        if n % d == 0:
            return False
    return True

def has_2_or_3(n):
    return '2' in str(n) or '3' in str(n)

def has_three_prime_factors(n):
    """Проверяет, что n = p1 * p2 * p3, где pi — простые (могут
повторяться)."""
    count = 0
    temp = n
    for p in range(2, math.isqrt(temp) + 1):
        while temp % p == 0:
            if not has_2_or_3(p):
                return False
            count += 1
            temp //= p
            if count > 3:
                return False
    if temp > 1:
        if not has_2_or_3(temp):
            return False
        count += 1
    return count == 3
```

```
found = 0
n = 5000001
while found < 5:
    if has_three_prime_factors(n):
        print(n)
        found += 1
    n += 1
```

Задача 4 (маска `1?6961*5`, делимость на 3013)

Условие: Числа ≤ 10^{10}, маска `1?6961*5`, делятся на 3013.

```
from itertools import product

results = []

# '?' — одна цифра
for d1 in '0123456789':
    prefix = f"1{d1}6961"
    # '*' — любая последовательность (включая пустую)
    # Число ≤ 10^10, prefix уже 6 цифр + '5' = 7, осталось до 3 цифр на
'*'
    for star_len in range(0, 4):
        if star_len == 0:
            num_str = prefix + '5'
            num = int(num_str)
            if num <= 10**10 and num % 3013 == 0:
                results.append(num)
        else:
            for star_digits in product('0123456789', repeat=star_len):
                star = ''.join(star_digits)
                num_str = prefix + star + '5'
                num = int(num_str)
                if num <= 10**10 and num % 3013 == 0:
                    results.append(num)

for r in sorted(set(results)):
    print(r)
```

Сводка: типовые приёмы в №25

Приём	Когда применять
Перебор до $\sqrt{n}$	Всегда при поиске делителей
`math.isqrt(n)`	Точный целочисленный корень
`is_prime()`	Проверка простоты
`product('01…9', repeat=k)`	Перебор `?` в маске
Перебор длины `*`	Маска с произвольной последовательностью
Факторизация (`while temp % p == 0`)	Разложение на простые множители

Домашнее задание

Задание 1

Найдите все числа от 100 000 до 200 000, у которых ровно 6 делителей.

Задание 2

Маска `2*4?5`. Найдите все числа ≤ 10^6, соответствующие маске и делящиеся на 15.

Задание 3

Найдите 5 чисел > 1 000 000, для которых максимальный простой делитель (не считая самого числа) оканчивается на 7 и не превышает 10 000.

Что мы сегодня изучили

Тема	Ключевые моменты
Делители до $\sqrt{n}$	`range(1, math.isqrt(n) + 1)`
Простота	`is_prime(n)` — перебор до $\sqrt{n}$
Маски	`?` → `product('0123456789', repeat=k)`, `*` → перебор длины
3 делителя	$n = p^2$, где p — простое
Факторизация	Деление `temp` на простые множители

> **Связь с ЕГЭ:** Задание №25 — 1 балл. Решается перебором с оптимизацией. Ключ — поиск делителей до $\sqrt{n}$.

На следующем занятии (Занятие 14)

Тема: Системы счисления и строки

- Системы счисления (углубление)
- Строковые алгоритмы (№12, 24)

Чек-лист занятия

- ☐ Умею находить делители перебором до $\sqrt{n}$
- ☐ Знаю разницу между `int(n**0.5)` и `math.isqrt(n)`
- ☐ Умею проверять число на простоту
- ☐ Умею перебирать маски с `?` и `*`
- ☐ Знаю, что число с 3 делителями = квадрат простого

- ☐ Выполнено домашнее задание

Глава 3. Системы счисления и строки

Занятие 14. Системы счисления (углубление)

План занятия

1. Позиционные СС: принцип и переводы
2. `int(s, base)`, `bin()`, `oct()`, `hex()`
3. Ручной перевод: деление с остатком
4. Алгоритмы с СС (№5 с усложнением)
5. Уравнения и перебор цифр/оснований (№14)
6. Практика

> **Цель занятия:** Свободно владеть переводами между системами счисления и решать задачи №5 и №14 любой сложности.

1. Позиционные системы счисления: принцип

Развёрнутая форма записи числа

Число $a_n a_{n-1} \dots a_1 a_0$ в системе с основанием b равно:

$$a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \dots + a_1 \cdot b + a_0$$

Пример: $123_5 = 1 \cdot 5^2 + 2 \cdot 5 + 3 = 25 + 10 + 3 = 38_{10}$

Стандартные СС

Основание	Название	Цифры
2	Двоичная	0, 1
8	Восьмеричная	0–7
10	Десятичная	0–9
16	Шестнадцатеричная	0–9, A–F
27	27-ричная	0–9, A–Q

Нестандартные СС (основание > 10)

Для оснований > 10 цифры обозначаются буквами: A=10, B=11, ..., Z=35. Для оснований > 36 нужны свои обозначения (в ЕГЭ не встречаются).

2. `int(s, base)`, `bin()`, `oct()`, `hex()`

Встроенные функции

Функция	Действие	Пример	Результат
`int(s, base)`	Из СС с основанием `base` в десятичную	`int('101', 2)`	`5`
`bin(n)`	Из десятичной в двоичную	`bin(42)`	`'0b101010'`
`oct(n)`	Из десятичной в восьмеричную	`oct(42)`	`'0o52'`
`hex(n)`	Из десятичной в 16-ричную	`hex(42)`	`'0x2a'`

Ограничения `int(s, base)`

- Основание `base` от 2 до 36.
- Цифры: 0–9, A–Z (регистр не важен).
- Если цифра ≥ основания — `ValueError`.

```
int('1A', 16)   # 26
int('1A', 10)   # ValueError! 'A' не цифра в десятичной
int('1G', 16)   # ValueError! 'G' не цифра в 16-ричной
```

3. Ручной перевод: деление с остатком

Из десятичной в любую СС

Делим число на основание, записываем остатки, переворачиваем.

```
def to_base(n, base):
    """Перевод числа n в СС с основанием base (2..36)."""
    if n == 0:
        return '0'
    digits = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    result = ''
    while n > 0:
        result = digits[n % base] + result
        n //= base
    return result

print(to_base(255, 16))  # 'FF'
print(to_base(255, 2))   # '11111111'
```

Из любой СС в десятичную (ручной способ)

```
def from_base(s, base):
    """Перевод строки s из СС с основанием base в десятичную."""
    digits = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    result = 0
    for char in s:
        result = result * base + digits.index(char.upper())
```

```
    return result

print(from_base('FF', 16))  # 255
```

> Ручные функции нужны для оснований > 36 или когда `int(s, base)` не работает.

4. Алгоритмы с СС (№5 с усложнением)

Шаблон для алгоритмов с переводом

```
def algo(N):
    # 1. Перевод в нужную СС
    s = to_base(N, BASE)  # или bin(N)[2:] для двоичной

    # 2. Обработка по условию
    if N % K == 0:
        s += s[-M:]  # Дописать последние M цифр
    else:
        remainder = N % K
        # ... обработка остатка ...
        s += to_base(remainder * K, BASE)

    # 3. Обратно в десятичную
    return int(s, BASE)
```

Типичные ошибки

- Забыть дописать разряды **в нужной СС**.
- Перепутать направление дописывания (слева/справа).
- Использовать `int(s, base)` без проверки, что `s` не пустая строка.

5. Уравнения и перебор цифр/оснований (№14)

Перебор неизвестной цифры

```
# Уравнение: 1x2_3 + 3×1_4 = 25, найти x
for x in range(0, 10):
    try:
        a = int(f"1{x}2", 3)
        b = int(f"3{x}1", 4)
        if a + b == 25:
            print(f"x = {x}")
    except ValueError:
        pass
```

Перебор основания

```
# 123_base = 38_10, найти основание
for base in range(4, 37):  # Основание > 3 (макс. цифра)
    try:
        if int('123', base) == 38:
            print(f"Основание: {base}")
```

```
    except ValueError:
        pass
```

Перебор двух переменных

```
# y04×5_11 + 253xy_8 — найти x, y, при которых сумма кратна 117
min_value = float('inf')
answer = None

for x in range(0, 8):   # x — цифра в 8-ричной (0..7)
    for y in range(0, 8):  # y — цифра в 8-ричной (0..7)
        try:
            a = int(f"{y}04{x}5", 11)
            b = int(f"253{x}{y}", 8)
            total = a + b
            if total % 117 == 0 and total < min_value:
                min_value = total
                answer = total // 117
        except ValueError:
            pass

print(answer)
```

6. Практика

Задача 1 (№5, двоичная запись, условное дописывание)

Условие:

Алгоритм получает на вход натуральное число N и строит по нему новое число R:

1. Строится двоичная запись числа N.
2. Если N делится на 3, то к двоичной записи справа дописываются три последние двоичные цифры.
3. Если N не делится на 3, то остаток от деления умножается на 3, переводится в двоичную запись и дописывается справа.
4. Результат переводится в десятичную систему.

Пример: N = 12 (1100_2). 12 делится на 3 → дописываем '100' → 1100100_2 = 100. N = 4 (100_2). 4 не делится на 3, остаток 1, 1×3=3 → 11_2 → 10011_2 = 19.

Вопрос: Укажите максимальное число R, не превышающее 137.

Код:

```
def algo(N):
    s = bin(N)[2:]  # Двоичная запись без '0b'
    if N % 3 == 0:
        s += s[-3:]  # Три последние цифры
    else:
        r = (N % 3) * 3
        s += bin(r)[2:]  # Остаток × 3 в двоичной
    return int(s, 2)
```

```
results = []
for N in range(1, 200):
    R = algo(N)
    if R <= 137:
        results.append(R)

print(max(results))
```

Задача 2 (№5, троичная запись, сумма цифр)

Условие:

Алгоритм для натурального числа N:

1. Строится троичная запись числа N.
2. Если N делится на 3, то к записи справа дописываются две последние троичные цифры.
3. Если N не делится на 3, то вычисляется сумма цифр троичной записи, эта сумма умножается на 3, переводится в троичную систему и дописывается справа.
4. Результат переводится в десятичную систему.

Пример: N = 8 = 22_3. 8 не кратно 3, сумма цифр = 4, 4×3=12=110_3 → 22110_3 = 228. N = 9 = 100_3. 9 кратно 3 → дописываем '00' → 10000_3 = 81.

Вопрос: Укажите число R, ближайшее к 826.

Код:

```
def to_base3(n):
    if n == 0:
        return '0'
    result = ''
    while n > 0:
        result = str(n % 3) + result
        n //= 3
    return result

def algo(N):
    s = to_base3(N)
    if N % 3 == 0:
        s += s[-2:]
    else:
        digit_sum = sum(int(d) for d in s)
        add = digit_sum * 3
        s += to_base3(add)
    return int(s, 3)

results = []
for N in range(1, 500):
    results.append(algo(N))

target = 826
closest = min(results, key=lambda r: abs(r - target))
print(closest)
```

Задача 3 (№5, удаление первой единицы и следующих нулей)

Условие:

Автомат обрабатывает натуральное число N:

1. Строится двоичная запись числа N.
2. Удаляется первая слева единица и все следующие непосредственно за ней нули. Если после этого в числе не остаётся цифр, результат считается равным нулю.
3. Полученное число переводится в десятичную запись.
4. Новое число вычитается из исходного, разность выводится на экран.

Пример: N = 11 (1011_2). Удаляем первую '1' и следующий за ней '0' → '11' → 3. 11 − 3 = 8.

Вопрос: Сколько разных значений будет показано на экране при вводе всех натуральных чисел от 10 до 1000?

Код:

```
def algo(N):
    s = bin(N)[2:]
    # Находим первую '1'
    idx = s.find('1')
    if idx == -1:
        after = '0'
    else:
        # Пропускаем первую '1' и все '0' сразу после неё
        i = idx + 1
        while i < len(s) and s[i] == '0':
            i += 1
        after = s[i:]
        if after == '':
            after = '0'
    new_num = int(after, 2) if after else 0
    return N - new_num

results = set()
for N in range(10, 1001):
    results.add(algo(N))

print(len(results))
```

Задача 4 (№5, восьмеричная запись, дописывание слева)

Условие:

Алгоритм для натурального числа N > 20:

1. Строится восьмеричная запись числа N.
2. Если N делится на 7, то к восьмеричной записи справа дописываются её последние две цифры.

3. Если N не делится на 7, то остаток от деления N на 7 умножается на 7, переводится в восьмеричную запись и приписывается **слева**.
4. Результат — число R в десятичной системе.

Пример: N = 21 (25_8). 21 кратно 7 → 2525_8 = 1365. N = 22 (26_8). 22 не кратно 7, остаток 1, $1×7=7=7_8$ → 726_8 = 470.

Вопрос: Укажите такое N, для которого R является наименьшим среди чисел, превышающих 500.

Код:

```
def algo(N):
    s = oct(N)[2:]
    if N % 7 == 0:
        s += s[-2:]
    else:
        r = (N % 7) * 7
        s = oct(r)[2:] + s  # Дописываем СЛЕВА
    return int(s, 8)

best_N = None
best_R = float('inf')

for N in range(21, 1000):
    R = algo(N)
    if R > 500 and R < best_R:
        best_R = R
        best_N = N

print(best_N)
```

Задача 5 (подсчёт цифр со значением > 9 в 27-ричной записи)

Условие:

Дано выражение: $2 \cdot 729^{2014} + 2 \cdot 81^{2018} + 2 \cdot 27^{2020} - 2 \cdot 9^{2022} - 2024$. Определите количество цифр с числовым значением, превышающим 9, в 27-ричной записи этого числа.

Вопрос: Сколько цифр со значением > 9 (т.е. букв A–Q, соответствующих 10–26)?

Код:

```
def to_base27(n):
    digits = '0123456789ABCDEFGHIJKLMNOPQ'
    result = ''
    while n > 0:
        result = digits[n % 27] + result
        n //= 27
    return result if result else '0'

value = 2 * 729**2014 + 2 * 81**2018 + 2 * 27**2020 - 2 * 9**2022 - 2024
s = to_base27(value)
# Цифры > 9: A(10), B(11), ... , Q(26)
count = sum(1 for c in s if c in 'ABCDEFGHIJKLMNOPQ')
print(count)
```

Задача 6 (подсчёт цифр 2 в троичной записи)

Условие:

Значение арифметического выражения $9^8 + 3^5 - 9$ записали в системе счисления с основанием 3. Сколько цифр 2 содержится в этой записи?

Код:

```
value = 9**8 + 3**5 - 9

def to_base3(n):
    result = ''
    while n > 0:
        result = str(n % 3) + result
        n //= 3
    return result if result else '0'

s = to_base3(value)
print(s.count('2'))
```

Задача 7 (две переменные в разных СС, кратность 117)

Условие:

Операнды арифметического выражения записаны в системах счисления с основаниями 11 и 8: $y04x5_{11} + 253xy_8$. Переменные x и y — допустимые в данных СС неизвестные цифры. Найдите значения x и y, при которых сумма минимальна и кратна 117. Вычислите частное от деления суммы на 117.

Код:

```
min_total = float('inf')
answer = None

for x in range(0, 8):   # x — цифра в 8-ричной СС (0..7)
    for y in range(0, 8):  # y — цифра в 8-ричной СС (0..7)
        try:
            a = int(f"{y}04{x}5", 11)
            b = int(f"253{x}{y}", 8)
            total = a + b
            if total % 117 == 0 and total < min_total:
                min_total = total
                answer = total // 117
        except ValueError:
            pass

print(answer)
```

Домашнее задание

Задание 1

Решите уравнение: $2x3_5 + 1x4_6 = 50$. Найдите цифру x.

Задание 2

Значение выражения $7^{2024} + 7^{1000} - 7$ записали в семеричной системе счисления. Сколько цифр 6 содержится в этой записи?

Задание 3

Алгоритм: строится двоичная запись N. Если N чётно — справа дописывается '10', иначе '01'. Найдите минимальное N, при котором результат R > 200.

Что мы сегодня изучили

Тема	Ключевые моменты
Встроенные функции	`int(s, base)`, `bin()`, `oct()`, `hex()`
Ручной перевод	`to_base(n, base)` — деление с остатком
Алгоритмы с СС	Дописывание разрядов справа и слева, обработка остатка
Перебор цифр	`for x in range(base)`, `int(f"...{x}...", base)`
Подсчёт цифр	`.count()`, проверка букв для основания > 10

> **Связь с ЕГЭ:** Задания №5 и №14 — 2 балла. Требуют свободного владения переводом между СС.

На следующем занятии (Занятие 15)

Тема: Строковые алгоритмы (задания №12, 24)

- Редактор строк: `while "подстрока" in s:`
- Скользящее окно
- Поиск максимальной подстроки

Чек-лист занятия

- ☐ Знаю `int(s, base)` и ограничение base ≤ 36
- ☐ Умею писать `to_base(n, base)` вручную
- ☐ Умею оформлять алгоритм с СС в функцию
- ☐ Умею перебирать неизвестные цифры и основания
- ☐ Знаю, как считать конкретные цифры в записи числа
- ☐ Выполнено домашнее задание

Занятие 15. Строковые алгоритмы (задания №12, 24)

План занятия

1. Задание №12: редактор строк, цикл `while` с заменами
2. Порядок замен и почему он важен
3. Задание №24: скользящее окно и два указателя
4. Поиск максимальной подстроки с ограничениями
5. Практика

> **Цель занятия:** Научиться решать задания №12 (редактор строк) и №24 (обработка строк из файла). Это 2 первичных балла.

1. Задание №12: редактор строк

Суть задания

Исполнитель Редактор получает строку и выполняет замены подстрок по циклу `ПОКА условие`.

Команды:

- `заменить(v, w)` — заменить **первое слева** вхождение `v` на `w`.
- `нашлось(v)` — проверить, есть ли `v` в строке.

Перевод на Python

```
s = "..."  # Исходная строка

while "111" in s or "222" in s:  # ПОКА нашлось(111) ИЛИ нашлось(222)
    s = s.replace("111", "22", 1)  # заменить(111, 22) — первое вхождение
    s = s.replace("222", "1", 1)   # заменить(222, 1)
```

`replace` с третьим аргументом

```
s.replace(old, new, count)
# count — сколько замен сделать (1 = только первое вхождение)
```

Порядок замен важен!

В цикле `ПОКА` обе замены выполняются **на каждой итерации**, в том порядке, в котором записаны. После первой замены строка меняется, и вторая замена работает уже с изменённой строкой.

```
# Правильно (как в условии):
while "111" in s or "222" in s:
    s = s.replace("111", "22", 1)  # Сначала эта
    s = s.replace("222", "1", 1)   # Потом эта

# Неправильно (другой порядок — другой результат):
while "111" in s or "222" in s:
```

```
s = s.replace("222", "1", 1)   # Сначала эта
s = s.replace("111", "22", 1)  # Потом эта
```

2. Порядок замен и почему он важен

Пример

Строка: `"111222"`

Вариант А (сначала 111→22):

1. `111222` → находим `111`, заменяем на `22` → `22222`
2. `22222` → находим `222`, заменяем на `1` → `122`
3. `122` → нет `111` и нет `222` → стоп.

Вариант Б (сначала 222→1):

1. `111222` → находим `222`, заменяем на `1` → `1111`
2. `1111` → находим `111`, заменяем на `22` → `221`
3. `221` → нет `111` и нет `222` → стоп.

Результаты разные!

Что делать на ЕГЭ

- Точно копируйте порядок команд из условия.
- Если замена не срабатывает (нет подстроки), `replace` с `count=1` просто возвращает исходную строку — это нормально.

3. Задание №24: скользящее окно и два указателя

Суть задания

Дан текстовый файл. Нужно найти максимальную подстроку, удовлетворяющую условиям:

- Не содержит определённых символов.
- Содержит определённые символы в нужном порядке.
- Имеет ограничения на количество разных символов.

Метод скользящего окна

Два указателя `left` и `right` движутся по строке:

- `right` расширяет окно вправо.
- Когда условие нарушается — `left` сдвигается вправо, сокращая окно.

```
s = open("24.txt").read().strip()
max_len = 0
```

```
left = 0

for right in range(len(s)):
    # Проверяем, удовлетворяет ли s[left:right+1] условию
    while условие_нарушено:
        left += 1
    max_len = max(max_len, right - left + 1)
```

Метод флагов и счётчиков

Для условий типа «символы X и Y не стоят рядом» или «ровно k вхождений подстроки»:

```
s = open("24.txt").read().strip()
max_len = 0
cur_len = 0

for char in s:
    if char in allowed:
        cur_len += 1
        max_len = max(max_len, cur_len)
    else:
        cur_len = 0
```

4. Поиск максимальной подстроки с ограничениями

Типовые ограничения

Условие	Подход
Нет символов из набора	Флаг: `if char in forbidden: cur = 0`
Только символы из набора	Флаг: `if char not in allowed: cur = 0`
Ровно k вхождений подстроки	Скользящее окно + счётчик вхождений
Все цифры встречаются	Счётчик цифр + скользящее окно
Гласная+гласная+согласная	Проверка троек подряд

Чтение файла

```
# Способ 1: весь файл в строку
with open("24.txt") as f:
    s = f.read().strip()

# Способ 2: по строкам (если файл многострочный)
with open("24.txt") as f:
    lines = [line.strip() for line in f]
```

5. Практика

Задача 1 (№12, замена 111 и 222)

Условие:

Программа:

```
ПОКА нашлось(111) ИЛИ нашлось(222)
    заменить(111, 22)
    заменить(222, 1)
КОНЕЦ ПОКА
```

Исходная строка содержит > 200 единиц и не содержит других цифр. После выполнения — тоже только единицы.

Вопрос: Какое наименьшее количество единиц могло быть в исходной строке?

Код:

```python
for n in range(201, 500):  # Перебираем длину исходной строки
    s = '1' * n
    while '111' in s or '222' in s:
        s = s.replace('111', '22', 1)
        s = s.replace('222', '1', 1)
    # Проверяем: состоит ли результат только из единиц?
    if set(s) == {'1'}:  # Все символы — единицы
        print(f"n = {n}")
        break
```

Задача 2 (№12, замена 1111 и 222)

Условие:

Программа:

```
ПОКА нашлось(1111)
    заменить(1111, 22)
    заменить(222, 1)
КОНЕЦ ПОКА
```

Исходная строка содержит > 200 единиц.

Вопрос: При какой наименьшей длине исходной строки результат будет содержать **наибольшее возможное** число единиц?

Код:

```python
max_ones = -1
best_n = None

for n in range(201, 500):
    s = '1' * n
    while '1111' in s:
        s = s.replace('1111', '22', 1)
        s = s.replace('222', '1', 1)
    ones = s.count('1')
    if ones > max_ones:
        max_ones = ones
        best_n = n
    elif ones == max_ones and n < best_n:
        best_n = n
```

```
print(f"Наименьшая длина: {best_n}")
print(f"Наибольшее число единиц: {max_ones}")
```

Задача 3 (№24, группы гласная+гласная+согласная)

Условие:

Текстовый файл содержит буквы A, C, D, F, O. Гласные: A, O. Согласные: C, D, F. Определите максимальное количество идущих подряд групп вида **гласная + гласная + согласная**.

Анализ: Группа — это ровно 3 символа: ГГС. Они должны идти подряд без перекрытия.

Код:

```
with open("24.txt") as f:
    s = f.read().strip()

vowels = set('AO')
consonants = set('CDF')

max_groups = 0
i = 0

while i ⩽ len(s) - 3:
    groups = 0
    while i ⩽ len(s) - 3 and s[i] in vowels and s[i+1] in vowels and
s[i+2] in consonants:
        groups += 1
        i += 3
    max_groups = max(max_groups, groups)
    i += 1

print(max_groups)
```

Сводка методов для №12 и №24

Задание	Метод	Ключевые инструменты
№12	Цикл `while` с заменами	`s.replace(old, new, 1)`, проверка `in`
№24 (нет символов)	Флаг + счётчик	`if char in forbidden: cur=0`
№24 (ровно k вхождений)	Скользящее окно	`left`, `right`, счётчик вхождений
№24 (все цифры)	Скользящее окно + словарь	`dict` для подсчёта уникальных
№24 (группы символов)	Пошаговый проход по 3	`while i ⩽ len(s)-3`

Домашнее задание

Задание 1 (№12)

Программа:

```
ПОКА нашлось(222) ИЛИ нашлось(888)
    ЕСЛИ нашлось(222)
        ТО заменить(222, 8)
        ИНАЧЕ заменить(888, 2)
КОНЕЦ ПОКА
```

Исходная строка из 100 цифр 8. Какая строка получится?

Задание 2 (№24)

Текстовый файл содержит буквы A, B, C. Найдите максимальную длину подстроки, в которой никакие две одинаковые буквы не стоят рядом.

Задание 3 (№24)

Текстовый файл содержит цифры. Найдите максимальную длину подстроки, в которой сумма цифр не превышает 100.

Что мы сегодня изучили

Тема	Ключевые моменты
№12: редактор	`while "v" in s: s = s.replace("v", "w", 1)`
Порядок замен	Важен! Меняет результат
№24: скользящее окно	`left`, `right`, счётчики
№24: группы символов	Проверка троек, проход с шагом
№24: ровно k вхождений	Скользящее окно + подсчёт подстроки

> **Связь с ЕГЭ:** Задания №12 и №24 — 2 балла. Требуют аккуратной работы со строками.

На следующем занятии (Занятие 16)

Тема: Рекурсия

- • Базовый и рекурсивный случай
- • Стек вызовов, `RecursionError`
- • `sys.setrecursionlimit()`

Чек-лист занятия

- ☐ Умею переписывать программу редактора на Python
- ☐ Понимаю, что `replace( ... , 1)` заменяет только первое вхождение
- ☐ Знаю, что порядок замен в цикле важен
- ☐ Умею применять скользящее окно для №24

- ☐ Умею считать вхождения подстроки при сдвиге окна
- ☐ Выполнено домашнее задание

Глава 4. Рекурсия

Занятие 16. Рекурсия: базовые принципы

План занятия

1. Что такое рекурсия: базовый и рекурсивный случай
2. Стек вызовов: как работает рекурсия
3. Ограничение глубины и `RecursionError`
4. Рекурсивные функции в задании №16
5. Рекурсивный подсчёт путей (задание №23)
6. Практика

> **Цель занятия:** Понять принцип рекурсии и научиться применять её для задач №16 и №23.

1. Что такое рекурсия: базовый и рекурсивный случай

Определение

Рекурсия — это приём, при котором функция вызывает саму себя для решения подзадачи меньшего размера.

Два обязательных компонента

1. **Базовый случай** — условие остановки. Без него рекурсия будет бесконечной.
2. **Рекурсивный случай** — вызов функции с новыми аргументами, приближающими к базовому случаю.

Пример: факториал

$n! = n \cdot (n-1) \cdot \dots \cdot 1$, $0! = 1$

```
def factorial(n):
    if n <= 1:          # Базовый случай
        return 1
    return n * factorial(n - 1)  # Рекурсивный случай

print(factorial(5))  # 120
```

Как это работает для `factorial(3)`:

```
factorial(3)
  → 3 * factorial(2)
       → 2 * factorial(1)
            → 1  (базовый случай)
```

```
        → 2 * 1 = 2
    → 3 * 2 = 6
```

Пример: сумма цифр числа

```python
def sum_of_digits(n):
    if n == 0:            # Базовый случай
        return 0
    return n % 10 + sum_of_digits(n // 10)  # Рекурсивный случай

print(sum_of_digits(12345))  # 15
```

2. Стек вызовов: как работает рекурсия

Что такое стек вызовов

При каждом вызове функции Python сохраняет её состояние (аргументы, локальные переменные, адрес возврата) в **стек вызовов** (call stack). Когда функция завершается, её состояние снимается со стека.

Визуализация для `factorial(3)`

```
Стек:

┌──────────────────┐
│ factorial(3)     │  ← верхушка стека
│ n = 3            │
│ ожидает 3 * ?    │
├──────────────────┤
│ factorial(2)     │
│ n = 2            │
│ ожидает 2 * ?    │
├──────────────────┤
│ factorial(1)     │
│ n = 1            │
│ возвращает 1     │  ← базовый случай, стек начинает «схлопываться»
└──────────────────┘
```

Почему рекурсия может быть медленной

Каждый вызов создаёт новый «слой» в стеке. Для `factorial(1000)` это 1000 слоёв — нормально. Но для `fibonacci(50)` без оптимизации — это миллионы вызовов из-за повторных вычислений.

3. Ограничение глубины и `RecursionError`

Стандартный лимит

Python ограничивает глубину рекурсии примерно **1000 вызовов**. Это защита от переполнения стека.

```python
def deep(n):
    if n == 0:
        return 0
    return deep(n - 1) + 1
```

```
# deep(2000)  # RecursionError: maximum recursion depth exceeded
```

Увеличение лимита

```
import sys
sys.setrecursionlimit(5000)  # Увеличиваем до 5000

def deep(n):
    if n == 0:
        return 0
    return deep(n - 1) + 1

print(deep(2000))  # Теперь работает
```

Опасность

Слишком большой лимит может привести к **крашу программы** (переполнение стека C). Для ЕГЭ хватает 5000–10000.

Альтернатива: итеративный подход

Если глубина > 10 000 — лучше переписать функцию без рекурсии (циклом).

4. Рекурсивные функции в задании №16

Типичная задача

Даны соотношения для F(n). Нужно вычислить F(N) для конкретного N или найти N по значению.

Шаблон

```
def F(n):
    if n == 0:          # Базовый случай из условия
        return 0
    if n % 4 < 2:       # Условие из задачи
        return F(n // 4) + n % 4
    else:
        return F(n // 4) + n % 4 - 1
```

Типичные ошибки

- Забыть базовый случай → бесконечная рекурсия.
- Перепутать `n % 4` (остаток) и `n // 4` (целая часть).
- Не учесть, что функция может зависеть от двух параметров: `F(n, m)`.

5. Рекурсивный подсчёт путей (задание №23)

Суть задачи

Исполнитель преобразует число на экране. Команды:

- Прибавить 1, 2, 3.
- Умножить на 2, 3.

Нужно найти **количество программ**, переводящих число A в число B.

Рекурсивное решение

```
def paths(start, end):
    if start == end:      # Достигли цели — одна программа
        return 1
    if start > end:       # Перелёт — тупик
        return 0
    return paths(start + 1, end) + paths(start * 2, end)
```

Ограничения на траекторию

«Содержит число X», «не содержит число Y» → добавляем флаги:

```
def paths(start, end, passed_required=False, hit_forbidden=False):
    if hit_forbidden:           # Зашли в запрещённое число
        return 0
    if start == end:
        return 1 if passed_required else 0  # Должны были пройти через X
    if start > end:
        return 0

    next_passed = passed_required or (start == REQUIRED_NUM)
    next_forbidden = hit_forbidden or (start == FORBIDDEN_NUM)

    return paths(start + 1, end, next_passed, next_forbidden) + \
           paths(start * 2, end, next_passed, next_forbidden)
```

6. Практика

Задача 1 (№16, система F и G)

Условие:

F(n) = F(n − 5) + 5580, если n ≥ 25; F(n) = 12 × (G(n − 11) − 14), если n < 25; G(n) = n / 6 + 34, если n ≥ 395 881; G(n) = 13 + G(n + 39), если n < 395 881.

Найти F(937).

Анализ:

- F(937): 937 ≥ 25 → F(937) = F(932) + 5580 = ... (много шагов до n < 25).

• G растёт при n < 395 881: G(n) = 13 + G(n+39). Рекурсия «вверх». Нужно дойти до n ≥ 395 881.

```
import sys
sys.setrecursionlimit(1000000)

def G(n):
    if n >= 395881:
        return n / 6 + 34  # Возвращает float!
    return 13 + G(n + 39)

def F(n):
    if n >= 25:
        return F(n - 5) + 5580
    return 12 * (G(n - 11) - 14)

print(F(937))
```

> **Внимание:** G возвращает `float` (деление `/`). В ЕГЭ обычно работают с целыми, но здесь формула дана с `/6`. Ответ может быть дробным.

Задача 2 (№16, факториал через рекурсию)

Условие:

F(1) = 1; F(n) = F(n − 1) · (n + 1), при n > 1.

Найти F(5).

```
def F(n):
    if n == 1:
        return 1
    return F(n - 1) * (n + 1)

print(F(5))
```

Ручная проверка: F(1) = 1 F(2) = 1 · 3 = 3 F(3) = 3 · 4 = 12 F(4) = 12 · 5 = 60 F(5) = 60 · 6 = 360

Задача 3 (№23, две команды)

Условие:

Исполнитель Удвоитель:

1. Прибавь 1.
2. Умножь на 2.

Сколько программ преобразуют 3 в 24?

```
def paths(start, end):
    if start == end:
        return 1
    if start > end:
        return 0
    return paths(start + 1, end) + paths(start * 2, end)
```

```
print(paths(3, 24))
```

Задача 4 (№23, содержит 8, не содержит 13)

Условие:

Исполнитель: +1, +2, ×3. Сколько программ из 3 в 18, содержащих 8 и не содержащих 13?

```
def paths(start, end, passed_8=False, hit_13=False):
    if hit_13:  # Зашли в запрещённое число
        return 0
    if start == end:
        return 1 if passed_8 else 0  # Должны были пройти через 8
    if start > end:
        return 0

    next_passed_8 = passed_8 or (start == 8)
    next_hit_13 = hit_13 or (start == 13)

    return paths(start + 1, end, next_passed_8, next_hit_13) + \
           paths(start + 2, end, next_passed_8, next_hit_13) + \
           paths(start * 3, end, next_passed_8, next_hit_13)

print(paths(3, 18))
```

Разбор флагов:

- `passed_8` — прошли ли уже через число 8.
- `hit_13` — зашли ли в запрещённое число 13.
- Флаги обновляются **до** вызова рекурсии.
- В конце (`start == end`) проверяем, что `passed_8 == True` .

Сводка: рекурсия в №16 и №23

Задание	Тип рекурсии	Особенности
№16	Прямая (дана формула)	Базовый случай из условия, `//` и `%`
№16 (система)	Взаимная (F → G, G → F)	Может расти «вверх» (G(n+39))
№23 (базовый)	Подсчёт путей	`if start==end: return 1`
№23 (с условиями)	С флагами	`passed_required` , `hit_forbidden`

Домашнее задание

Задание 1 (№16)

F(1) = 1; F(n) = F(n−1) + 2n − 1, при n > 1. Найдите F(10).

Задание 2 (№16)

F(0) = 1; F(n) = F(n//2) + F(n//3), при n > 0. Найдите количество различных значений F(n) для n от 0 до 1000.

Задание 3 (№23)

Исполнитель: +1, ×2. Сколько программ из 2 в 40, не содержащих числа 10?

Что мы сегодня изучили

Тема	Ключевые моменты
Базовый случай	Условие остановки рекурсии
Стек вызовов	Хранение состояний, LIFO
`RecursionError`	Лимит ~1000, `sys.setrecursionlimit()`
№16: F(n)	Переписываем соотношения в `def F(n)`
№23: подсчёт путей	`if start==end: return 1`, флаги для условий

> **Связь с ЕГЭ:** Рекурсия — фундамент для заданий №16, 23. Следующее занятие добавит мемоизацию (`@lru_cache`).

На следующем занятии (Занятие 17)

Тема: Мемоизация и `@lru_cache` (задание №16)

- Кэширование результатов
- `@lru_cache(maxsize=None)`
- Когда кэш не спасает

Чек-лист занятия

- ☐ Понимаю разницу между базовым и рекурсивным случаем
- ☐ Знаю, что такое стек вызовов
- ☐ Умею увеличивать лимит рекурсии
- ☐ Могу написать рекурсивную функцию по соотношениям из №16
- ☐ Умею считать количество программ с условиями (№23)
- ☐ Выполнено домашнее задание

Занятие 17. Мемоизация и @lru_cache (задания №16, 23)

План занятия

1. Проблема повторных вычислений в рекурсии
2. Кэширование вручную: словарь `memo = {}`
3. `@lru_cache` — автоматическое кэширование
4. Когда кэш не спасает: слишком много состояний
5. Практика

> **Цель занятия:** Научиться ускорять рекурсивные функции с помощью мемоизации. Ключевой навык для №16 и №23.

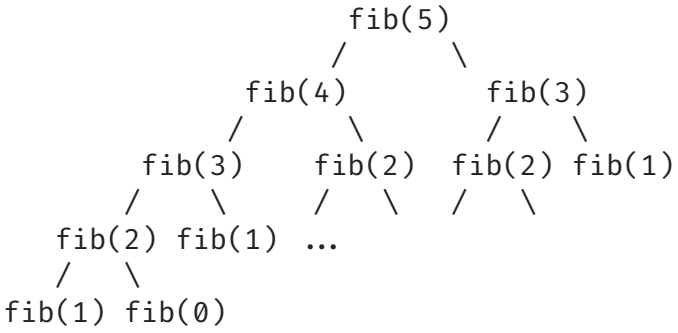
1. Проблема повторных вычислений в рекурсии

Дерево вызовов Фибоначчи

Рассмотрим рекурсивное вычисление чисел Фибоначчи:

```python
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
```

Для `fib(5)` дерево вызовов выглядит так:



Проблема: `fib(3)` вычисляется **дважды**, `fib(2)` — **трижды**. Количество вызовов растёт экспоненциально: $O(2^n)$.

```
# fib(35) без кэша — несколько секунд
# fib(50) без кэша — практически бесконечно
```

Решение: запоминать уже вычисленные результаты

Это называется **мемоизация** (от англ. *memoization* — запоминание).

2. Кэширование вручную: словарь `memo = {}`

Ручная мемоизация

```
memo = {}  # Словарь для хранения результатов

def fib(n):
    if n in memo:        # Проверяем, вычисляли ли уже
        return memo[n]
    if n <= 1:
        return n
    result = fib(n - 1) + fib(n - 2)
    memo[n] = result     # Сохраняем в кэш
    return result

print(fib(100))  # Мгновенно!
```

Ручная мемоизация для функции с двумя параметрами

```
memo = {}

def F(n, m):
    key = (n, m)          # Ключ — кортеж параметров
    if key in memo:
        return memo[key]
    if n == 0:
        return m
    result = F(n - 1, m) + F(n, m - 1)
    memo[key] = result
    return result
```

Недостатки ручного кэширования

- Громоздкий код: нужно писать проверку и сохранение для каждой функции.
- Кэш — глобальная переменная, может быть случайно изменена.
- Для нескольких функций нужно несколько словарей.

3. `@lru_cache` — автоматическое кэширование

Синтаксис

```
from functools import lru_cache

@lru_cache(maxsize=None)  # Декоратор над функцией
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
```

Что делает `@lru_cache`:

- Автоматически создаёт словарь-кэш для функции.

- Перед вызовом проверяет, есть ли результат для таких аргументов.
- Если есть — возвращает из кэша, не вычисляя заново.
- Если нет — вычисляет, сохраняет в кэш и возвращает.

Сравнение скорости

```
from functools import lru_cache
import time

# Без кэша
def fib_slow(n):
    if n <= 1:
        return n
    return fib_slow(n - 1) + fib_slow(n - 2)

# С кэшем
@lru_cache(maxsize=None)
def fib_fast(n):
    if n <= 1:
        return n
    return fib_fast(n - 1) + fib_fast(n - 2)

# fib_slow(35) — несколько секунд
# fib_fast(100) — мгновенно
```

`maxsize` — ограничение размера кэша

```
@lru_cache(maxsize=128)  # Хранить только 128 последних результатов
@lru_cache(maxsize=None)  # Хранить все результаты (для ЕГЭ — нормально)
```

Для функций с несколькими параметрами

```
@lru_cache(maxsize=None)
def F(n, m):
    if n == 0:
        return m
    return F(n - 1, m) + F(n, m - 1)

# Работает автоматически: кэш хранит результаты для каждой пары (n, m)
```

4. Когда кэш не спасает: слишком много уникальных состояний

Проблема большого кэша

Если функция вызывается с **миллионами разных аргументов**, кэш занимает слишком много памяти.

Пример:

```
@lru_cache(maxsize=None)
def F(n):
    if n <= 1:
        return 1
    return F(n - 1) + F(n // 2)
```

```
# F(1000000) — кэш будет содержать ~1 млн записей
# Может занять сотни мегабайт памяти
```

Когда это критично для ЕГЭ

- **№16:** Обычно глубина рекурсии ≤ 1000, кэш маленький — `@lru_cache` отлично работает.
- **№23:** Количество траекторий до 100, состояний мало — `@lru_cache` идеален.
- **Если n > 10^5 и цепочка вызовов длинная** — лучше использовать **восходящее ДП** (цикл вместо рекурсии, занятие 24).

Признаки, что кэш не справится

Признак	Что делать
n до 10^6 и более	Восходящее ДП (цикл)
Два параметра, каждый до 10^3	Кэш на 10^6 записей — ещё терпимо
Рекурсия с `n+1` (растёт вверх)	Осторожно: может уйти в бесконечность

5. Практика

Задача 1 (№16, факториал через рекурсию)

Условие:

F(1) = 1; F(n) = n · F(n − 1), при n > 1.

Найти F(2023) / F(2020).

Анализ: F(n) = n! (факториал). F(2023) / F(2020) = 2023! / 2020! = 2021 · 2022 · 2023.

Но для демонстрации глубины рекурсии решим рекурсивно:

```python
import sys
sys.setrecursionlimit(100000)
def F(n):
    if n == 1:
        return 1
    return n * F(n - 1)

print(F(2023) // F(2020))
```

Задача 2 (№16, система F и G)

Условие:

F(n) = 2 · (G(n − 3) + 8); G(n) = 2 · n, если n < 10; G(n) = G(n − 2) + 1, если n ≥ 10.

Найти F(15548).

Анализ:

- Глубокая рекурсия, возможен `RecursionError`.

Код:

```
import sys
sys.setrecursionlimit(20000)

def G(n):
    if n < 10:
        return 2 * n
    return G(n - 2) + 1

def F(n):
    return 2 * (G(n - 3) + 8)

print(F(15548))
```

Задача 3 (№23)

Условие:

Исполнитель: +1, +2, +3. Сколько программ из 1 в 35, содержащих число 7?

Код с мемоизацией:

```
from functools import lru_cache

@lru_cache(maxsize=None)
def paths(start, end, passed_7=False):
    if start == end:
        return 1 if passed_7 else 0
    if start > end:
        return 0

    next_passed = passed_7 or (start == 7)

    return (paths(start + 1, end, next_passed) +
            paths(start + 2, end, next_passed) +
            paths(start + 3, end, next_passed))

print(paths(1, 35))
```

Что даёт кэш: `paths(start, passed_7)` вызывается для каждого `start` от 1 до 35 и двух значений флага — всего ~70 уникальных состояний. Каждое вычисляется один раз. Без кэша — экспоненциальный взрыв.

Ответ: 5 906 209.

Сводка: ручной кэш vs @lru_cache

Критерий	Ручной `memo = {}`	`@lru_cache`
Код	Громоздкий	Одна строка
Ошибки	Легко забыть сохранить	Автоматически
Несколько параметров	Ключ — кортеж	Автоматически
Очистка кэша	`memo.clear()`	`func.cache_clear()`
Рекомендация для ЕГЭ	Только если забыли импорт	**Всегда использовать**

Домашнее задание

Задание 1 (№16)

F(0) = 1; F(n) = F(n // 2) + F(n // 3), при n > 0. Найдите количество **различных** значений F(n) для n от 0 до 1000.

Задание 2 (№16)

F(1) = 1; F(n) = F(n − 1) + 2n − 1, при n > 1. Найдите F(100). Сравните с формулой арифметической прогрессии.

Задание 3 (№23)

Исполнитель: +1, ×2. Сколько программ из 2 в 40, **не содержащих** числа 10?

Что мы сегодня изучили

Тема	Ключевые моменты
Повторные вычисления	Дерево вызовов, экспоненциальный рост
Ручной кэш	`memo = {}`, проверка `if key in memo`
`@lru_cache`	Одна строка над функцией, `maxsize=None`
Ограничения кэша	Слишком много состояний → восходящее ДП
Применение	№16 (рекурсивные функции), №23 (подсчёт путей)

> **Связь с ЕГЭ:** `@lru_cache` — обязательный инструмент для №16 и №23. Без него многие задачи решаются неприемлемо долго.

На следующем занятии (Занятие 18)

Тема: Рекурсивный перебор (задание №8, рекурсивный метод)

- Рекурсия для генерации комбинаций
- Обход дерева вариантов

- Когда рекурсивный перебор лучше `itertools`

Чек-лист занятия

- ☐ Понимаю, почему рекурсия без кэша медленная
- ☐ Умею писать ручной кэш со словарём
- ☐ Знаю синтаксис `@lru_cache(maxsize=None)`
- ☐ Понимаю, когда кэш не помогает (слишком много состояний)
- ☐ Применил `@lru_cache` к задачам №16 и №23
- ☐ Выполнено домашнее задание

Занятие 18. Рекурсивный перебор (задание №8, рекурсивный метод)

План занятия

1. Рекурсия для генерации комбинаций
2. Параметры состояния: позиция, последний символ, счётчики
3. Когда рекурсивный перебор лучше `itertools`
4. Кэширование промежуточных результатов
5. Практика

> **Цель занятия:** Освоить рекурсивный перебор с фильтрацией для задач №8, где `itertools.product` справляется, но рекурсия даёт больше гибкости и готовит к ДП.

1. Рекурсия для генерации комбинаций

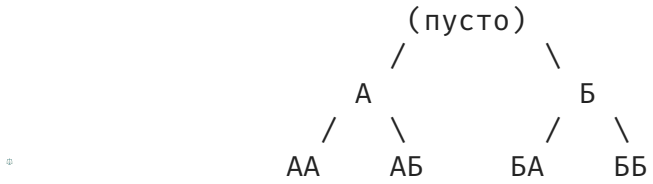
Идея

Вместо `itertools.product` можно строить слова рекурсивно: на каждом шаге выбираем одну букву и переходим к следующей позиции.

```
def generate(pos, word):
    if pos == L:                # Достигли нужной длины
        print(word)             # Обрабатываем слово
        return
    for char in alphabet:       # Перебираем буквы
        generate(pos + 1, word + char)
```

Обход дерева вариантов

Для алфавита `{А, Б}` и длины 2:



Рекурсия обходит это дерево **в глубину**: сначала левая ветка до конца, потом возвращается и идёт в правую.

Рекурсия vs `itertools.product`

Критерий	**`itertools.product`**	**Рекурсия**
Компактность	Одна строка	6–8 строк
Фильтрация	После генерации	Во время генерации (отсечение)
Доп. параметры	Нет	Можно передавать счётчики, флаги
Кэширование	Нет	Можно (`@lru_cache`)

> **Вывод:** Для простых задач используйте `product`. Для задач со сложными условиями или когда нужно ДП — используйте рекурсию.

2. Параметры состояния: позиция, последний символ, счётчики

Типовые параметры

Параметр	Назначение	Пример использования
`pos`	Текущая позиция (сколько букв уже поставлено)	`if pos == L: return 1`
`last`	Последняя поставленная цифра/буква	Проверка «не равны»
`count_X`	Сколько раз встретилась буква X	«Ровно одна цифра 8»
`has_Y`	Встречалась ли буква Y	«Хотя бы одна цифра 5»

Шаблон рекурсивной функции с параметрами

```
def count(pos, last, cnt_8, has_5):
    if pos == L:  # База: достигли нужной длины
        return 1 if условие_выполнено else 0

    total = 0
    for d in digits:
        if ограничение_на_соседей(last, d):
            new_cnt_8 = cnt_8 + (1 if d == 8 else 0)
            new_has_5 = has_5 or (d == 5)
            total += count(pos + 1, d, new_cnt_8, new_has_5)
    return total
```

3. Когда рекурсивный перебор лучше `itertools`

Преимущества рекурсии

1. **Раннее отсечение:** Если условие нарушено на 3-й позиции, мы не перебираем оставшиеся позиции.
2. **Сложные счётчики:** «Ровно две цифры со значением > 12, стоящие рядом» — легко отслеживать в параметрах.
3. **Кэширование:** Если функция зависит только от `(pos, last, счётчики)`, можно применить `@lru_cache`.

Когда `itertools` проще

- Объём перебора небольшой (до 10^6).
- Условия простые (`count` , `find`).
- Не нужно кэширование.

4. Кэширование промежуточных результатов

Когда кэширование помогает

Если рекурсивная функция зависит **только** от параметров состояния, а не от полной строки, можно применить `@lru_cache`.

```
from functools import lru_cache

@lru_cache(maxsize=None)
def count(pos, last):
    if pos == L:
        return 1
    total = 0
    for d in digits:
        if d ≠ last:  # Не равны
            total += count(pos + 1, d)
    return total
```

Здесь `count(5, 3)` вызывается много раз из разных веток. Кэш запоминает результат и не пересчитывает.

Когда кэширование НЕ помогает

Если параметры **уникальны** для каждой ветки (например, полная строка), кэш бесполезен.

5. Практика

Задача 1 (ровно одна цифра 8, нет одинаковых соседей)

Условие:

Сколько 15-ричных четырёхзначных чисел, содержащих ровно одну цифру 8, в которых никакие две одинаковые цифры не стоят рядом?

Анализ:

- 15-ричная система: цифры 0..14 (0..9, A..E).
- Четырёхзначное → первая цифра не 0.
- Рекурсивный перебор: $15^4 = 50625$ — можно и `product`, но для тренировки рекурсия.

```
L = 4
base = 15

def count(pos, last, cnt_8):
    if pos == L:
        return 1 if cnt_8 == 1 else 0

    total = 0
    for d in range(base):
        if pos == 0 and d == 0:  # Первая цифра не 0
```

```
            continue
        if d == last:  # Одинаковые не рядом
            continue
        new_cnt = cnt_8 + (1 if d == 8 else 0)
        if new_cnt > 1:  # Больше одной 8 — не подходит
            continue
        total += count(pos + 1, d, new_cnt)
    return total

print(count(0, -1, 0))
```

Параметры:

- `pos` — позиция (0..3).
- `last` — последняя цифра (для проверки повторов). Начало: -1 (любая цифра подойдёт).
- `cnt_8` — сколько цифр 8 уже поставлено.

Задача 2 (все цифры различны, чёт/нечет чередуются)

Условие:

Сколько пятизначных 8-ричных чисел, в которых все цифры различны и никакие две чётные или две нечётные не стоят рядом?

Анализ:

- Пятизначное → первая цифра не 0.
- 8-ричная: цифры 0..7.
- Чётные: 0, 2, 4, 6. Нечётные: 1, 3, 5, 7.
- Все цифры различны → нужно отслеживать использованные.

```
L = 5
base = 8

def count(pos, last, used_mask):
    if pos == L:
        return 1

    total = 0
    for d in range(base):
        if pos == 0 and d == 0:
            continue
        if used_mask & (1 << d):  # Цифра уже использована
            continue
        if last ≠ -1 and (d % 2) == (last % 2):  # Одинаковая чётность
            continue
        total += count(pos + 1, d, used_mask | (1 << d))
    return total

print(count(0, -1, 0))
```

used_mask — битовая маска использованных цифр. `1 << d` — бит для цифры d. Проверка: `used_mask & (1 << d)`.

Задача 3 (ровно одна гласная, ровно 1 раз)

Условие:

Вася составляет 5-буквенные слова из букв З, И, М, А. В каждом слове ровно одна гласная буква, и она встречается ровно 1 раз. Согласные могут повторяться.

Анализ:

- Буквы: З, И, М, А.
- Гласные: И, А.
- Согласные: З, М.
- Ровно одна гласная, ровно 1 раз → она одна и не повторяется.
- Это $4^5 = 1024$ варианта — `product` тоже справится. Но рекурсия показывает идею.

```
L = 5
letters = 'ЗИМА'
vowels = set('ИА')

def count(pos, vowel_count, which_vowel):
    """
    vowel_count — сколько гласных уже поставлено.
    which_vowel — какая именно гласная (None, 'И', 'А').
    """
    if pos == L:
        return 1 if vowel_count == 1 else 0

    total = 0
    for ch in letters:
        if ch in vowels:
            # Нельзя ставить другую гласную, если уже есть
            if vowel_count == 1:
                continue
            # Нельзя ставить ту же гласную повторно
            if which_vowel == ch:
                continue
            total += count(pos + 1, 1, ch)
        else:
            total += count(pos + 1, vowel_count, which_vowel)
    return total

print(count(0, 0, None))
```

Аналитическая проверка:

- Выбрать позицию для гласной: 5 способов.
- Выбрать гласную: 2 способа (И или А).
- Остальные 4 позиции — согласные: $2^4 = 16$ способов.
- Итого: $5 \times 2 \times 16 = 160$.

Сводка: параметры состояния

Параметр	Тип	Пример
`pos`	`int`	Текущая позиция (0..L-1)
`last`	`int` / `str`	Последний символ (или -1 в начале)
`cnt_X`	`int`	Счётчик символов X
`has_X`	`bool`	Флаг наличия символа X
`used_mask`	`int`	Битовая маска использованных цифр
`prev_high`	`bool`	Была ли предыдущая цифра с особым свойством

Домашнее задание

Задание 1

Сколько 6-буквенных слов из букв А, Б, В, в которых ровно две буквы А и они не стоят рядом?

Задание 2

Сколько 4-значных 7-ричных чисел, в которых цифры идут строго по возрастанию?

Задание 3

Сколько 5-буквенных слов из букв К, О, Т, в которых нет подстроки «ОО» и ровно одна буква О?

Что мы сегодня изучили

Тема	Ключевые моменты
Рекурсивная генерация	Строим слова посимвольно, обходя дерево вариантов
Параметры состояния	`pos` , `last` , счётчики, флаги, битовые маски
Раннее отсечение	Прекращаем ветку, если условие уже нарушено
Кэширование	`@lru_cache` для функций, зависящих от состояния

> **Связь с ЕГЭ:** Рекурсивный перебор — мост к ДП. Когда `product` не справляется (занятие 25), мы используем рекурсию с кэшем.

На следующем занятии (Занятие 19)

Тема: Рекурсия в подсчёте путей (задание №23, базовое)

- База: `start == target → 1`
- Переходы: `+1` , `*2`
- `@lru_cache` для ускорения

Чек-лист занятия

- ☐ Понимаю, как работает рекурсивный обход дерева вариантов
- ☐ Умею добавлять параметры состояния (счётчики, флаги)
- ☐ Знаю, когда рекурсия лучше `product`
- ☐ Умею применять `@lru_cache` к рекурсивному перебору
- ☐ Выполнено домашнее задание

Занятие 19. Рекурсия в подсчёте путей (задание №23, базовое)

План занятия

1. Суть задания №23: количество траекторий
2. Базовый шаблон рекурсивной функции
3. Добавление условий: содержит/не содержит число
4. `@lru_cache` для ускорения
5. Практика

> **Цель занятия:** Научиться считать количество программ-траекторий рекурсивно с мемоизацией. Это 1 первичный балл.

1. Суть задания №23: количество траекторий

Что дано

Исполнитель преобразует число на экране. Команды:

- Прибавить 1, 2, 3...
- Умножить на 2, 3...

Вопрос: Сколько существует **программ** (последовательностей команд), переводящих число A в число B?

Что такое траектория

Траектория — последовательность **всех промежуточных результатов** выполнения команд.

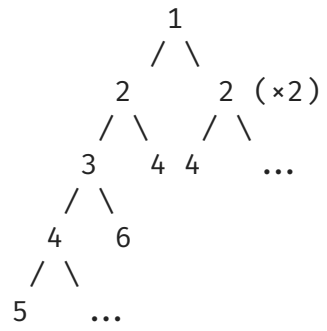
Пример: программа `+1, +2, ×2` для числа 3:

```
3 → 4 → 6 → 12
```

Траектория: `[3, 4, 6, 12]`.

Дерево вариантов

Для команд `+1` и `×2` из 1 в 5:



Каждый путь от корня до числа 5 — это одна программа.

2. Базовый шаблон рекурсивной функции

Рекурсивная формула

Пусть `paths(start, end)` — количество программ из `start` в `end`.

- Если `start == end` — мы уже в цели. Одна программа (пустая). Возвращаем 1.
- Если `start > end` — перелёт. Возвращаем 0.
- Иначе: сумма путей из всех возможных следующих чисел.

```
def paths(start, end):
    if start == end:
        return 1
    if start > end:
        return 0
    return paths(start + 1, end) + paths(start * 2, end)
```

Проблема: экспоненциальный взрыв

Для `end = 100` функция вызовет себя миллионы раз. Нужен **кэш**.

```
from functools import lru_cache

@lru_cache(maxsize=None)
def paths(start, end):
    if start == end:
        return 1
    if start > end:
        return 0
    return paths(start + 1, end) + paths(start * 2, end)
```

С кэшем `paths(start, end)` вычисляется **один раз** для каждого `start`. Сложность O(end).

3. Добавление условий: содержит/не содержит число

Условие «содержит число X»

Траектория **обязательно** должна проходить через X.

Решение: Разбиваем на две задачи:

1. Количество путей из A в X.
2. Количество путей из X в B.

Общее количество = `paths(A, X) * paths(X, B)`.

```
result = paths(1, 7) * paths(7, 35)
```

Условие «не содержит число Y»

Траектория **не должна** проходить через Y.

Решение: Добавляем в рекурсию проверку — если `start == Y`, возвращаем 0.

```
@lru_cache(maxsize=None)
def paths(start, end, forbidden):
    if start == end:
        return 1
    if start > end or start == forbidden:
        return 0
    return paths(start + 1, end, forbidden) + paths(start * 2, end, forbid
den)
```

Комбинация: содержит X и не содержит Y

Разбиваем на два этапа:

1. Из A в X (не заходя в Y).
2. Из X в B (не заходя в Y).

```
result = paths(A, X, Y) * paths(X, B, Y)
```

4. `@lru_cache` для ускорения

Сравнение с кэшем и без

end	Без кэша	С кэшем
20	~1 мс	~0.1 мс
30	~100 мс	~0.1 мс
50	Несколько секунд	~0.1 мс
100	Бесконечно	~0.1 мс

Когда кэша достаточно

Для задач №23 `end` редко превышает 100. Кэш на 100 состояний — это ничто.

Когда кэш НЕ нужен

Если задача решается разбиением (содержит X) — можно обойтись без рекурсии, используя ДП снизу вверх (итеративно). Об этом в занятии 26.

5. Практика

Задача 1 (содержит 14, не содержит 8)

Условие:

Исполнитель: +1, +2, ×2. Сколько программ из 3 в 18, содержащих 14 и не содержащих 8?

Анализ:

- Разбиваем на два этапа: 3 → 14 и 14 → 18.
- На обоих этапах нельзя заходить в 8.

```
from functools import lru_cache

@lru_cache(maxsize=None)
def paths(start, end, forbidden):
    if start == end:
        return 1
    if start > end or start == forbidden:
        return 0
    return (paths(start + 1, end, forbidden) +
            paths(start + 2, end, forbidden) +
            paths(start * 2, end, forbidden))

# Из 3 в 14, не заходя в 8
to_14 = paths(3, 14, 8)
# Из 14 в 18, не заходя в 8
from_14 = paths(14, 18, 8)

print(to_14 * from_14)
```

Почему умножаем: Каждая программа из 3 в 14 комбинируется с каждой программой из 14 в 18. Общее количество = произведение.

Задача 2 (содержит 8)

Условие:

Исполнитель Осень16: +1, +2, +4. Сколько программ из 1 в 15, содержащих 8?

Анализ:

- Разбиваем: 1 → 8 и 8 → 15.
- Запретов нет.

```
from functools import lru_cache

@lru_cache(maxsize=None)
def paths(start, end):
    if start == end:
        return 1
    if start > end:
        return 0
    return (paths(start + 1, end) +
            paths(start + 2, end) +
            paths(start + 4, end))

print(paths(1, 8) * paths(8, 15))
```

Задача 3 (без условий, три команды)

Условие:

Исполнитель Тренер: +1, +2. Сколько программ из 1 в 11?

```
from functools import lru_cache

@lru_cache(maxsize=None)
def paths(start, end):
    if start == end:
        return 1
    if start > end:
        return 0
    return paths(start + 1, end) + paths(start + 2, end)

print(paths(1, 11))
```

Аналитическая проверка: Это числа Фибоначчи. Количество программ из 1 в n = F(n-1), где F(1)=1, F(2)=1. Из 1 в 11: F(10) = 89.

Сводка: приёмы для №23

Условие	Метод
Нет условий	`paths(A, B)` — базовая рекурсия
Содержит X	`paths(A, X) * paths(X, B)`
Не содержит Y	Добавить `forbidden` в параметры
Содержит X и не содержит Y	`paths(A, X, Y) * paths(X, B, Y)`
Несколько запретов	Передать кортеж `forbidden`

Домашнее задание

Задание 1

Исполнитель: +1, ×2. Сколько программ из 2 в 20, не содержащих 10?

Задание 2

Исполнитель: +1, +3, ×2. Сколько программ из 1 в 25, содержащих 7?

Задание 3

Исполнитель: +1, +2, +3. Сколько программ из 2 в 30, содержащих 15 и не содержащих 10?

Что мы сегодня изучили

Тема	Ключевые моменты
Базовый шаблон	`if start==end: return 1; if start>end: return 0`
Кэширование	`@lru_cache` — обязательно для скорости
Содержит X	Разбиение на два этапа, перемножение
Не содержит Y	Добавление `forbidden` в параметры
Комбинация условий	Разбиение + запрет на обоих этапах

> **Связь с ЕГЭ:** Задание №23 — 1 балл. С `@lru_cache` решается за минуту.

Чек-лист занятия

- ☐ Знаю базовый шаблон рекурсивной функции для №23
- ☐ Понимаю, зачем нужен `@lru_cache`
- ☐ Умею решать задачи с условием «содержит X»
- ☐ Умею решать задачи с условием «не содержит Y»
- ☐ Выполнено домашнее задание

Занятие 20. Параллельные процессы (задание №22)

План занятия

1. Модель: процессы, зависимости, параллельное выполнение
2. Рекурсивный подсчёт времени завершения процесса
3. Кэширование для одинаковых подзадач

4. Практика

> **Цель занятия:** Научиться решать задание №22 — параллельные процессы с зависимостями. Это 1 первичный балл.

1. Модель: процессы, зависимости, параллельное выполнение

Суть задачи

Дано N процессов. У каждого есть:

- **ID** — идентификатор.
- **Время выполнения** (мс).
- **Зависимости** — список ID процессов, от которых он зависит.

Правила:

- Независимые процессы могут выполняться **параллельно**.
- Процесс B, зависящий от A, начинает выполняться только после **завершения всех** процессов, от которых он зависит.

Нужно найти:

- Минимальное время завершения **всех** процессов.
- Или сумму ID процессов, не завершившихся к моменту T.

Представление в виде графа

Процессы — вершины графа. Зависимость B → A означает, что B зависит от A (стрелка от A к B).

```
1 (4 мс) ──→ 3 (1 мс) ──→ 4 (7 мс)
2 (3 мс) ──┘
```

Время завершения процесса = его время выполнения + **максимум** из времён завершения процессов, от которых он зависит.

Формула

Пусть `T[i]` — минимальное время завершения процесса i.

```
T[i] = time[i] + max(T[j] для всех j, от которых зависит i)
```

Если процесс независимый (зависимостей нет), `max( ... )` = 0.

2. Рекурсивный подсчёт времени завершения процесса

Рекурсивная функция

```
def completion_time(pid):
    if pid in cache:
```

```
        return cache[pid]

    deps = dependencies[pid]  # Список ID процессов, от которых зависит pid
    if not deps:  # Независимый процесс
        cache[pid] = times[pid]
        return times[pid]

    max_dep_time = max(completion_time(dep) for dep in deps)
    cache[pid] = times[pid] + max_dep_time
    return cache[pid]
```

Почему рекурсия

Процессы образуют **направленный ациклический граф** (DAG). Рекурсия идёт по зависимостям вглубь до независимых процессов, а затем возвращает время наверх.

Опасность: циклы в зависимостях

В задачах ЕГЭ циклов практически нет, но если бы были — рекурсия зациклилась бы. Кэш спасает от повторных вычислений, но не от бесконечной рекурсии.

3. Кэширование для одинаковых подзадач

Зачем нужен кэш

Один процесс может зависеть от нескольких, а те — от одних и тех же «общих» процессов. Без кэша время завершения общего процесса вычислится многократно.

```
cache = {}  # pid → время завершения
```

Альтернатива: итеративное ДП (топологическая сортировка)

Если глубина рекурсии большая, можно обойти граф в порядке топологической сортировки. Но для ЕГЭ глубина редко превышает 1000.

```
# Итеративный подход:
for pid in topological_order:
    if dependencies[pid]:
        T[pid] = times[pid] + max(T[dep] for dep in dependencies[pid])
    else:
        T[pid] = times[pid]
```

4. Практика

Задача 1 (минимальное время завершения всех процессов)

Условие:

Дан файл с таблицей: ID, время, зависимости. Найти минимальное время завершения всей совокупности процессов.

Формат файла:

```
1	4	0
2	3	0
3	1	1;2
4	7	3
```

```
# Читаем файл
times = {}
dependencies = {}

with open("22.txt") as f:
    for line in f:
        parts = line.strip().split("\t")
        pid = int(parts[0])
        time = int(parts[1])
        deps_str = parts[2]

        times[pid] = time
        if deps_str == "0":
            dependencies[pid] = []
        else:
            dependencies[pid] = [int(x) for x in deps_str.split(";")]

# Рекурсивный подсчёт времени завершения
cache = {}

def completion_time(pid):
    if pid in cache:
        return cache[pid]

    deps = dependencies[pid]
    if not deps:
        cache[pid] = times[pid]
    else:
        max_dep = max(completion_time(dep) for dep in deps)
        cache[pid] = times[pid] + max_dep
    return cache[pid]

# Время завершения всех процессов — максимум по всем
total_time = max(completion_time(pid) for pid in times)
print(total_time)
```

Задача 2 (сумма ID незавершённых к моменту T)

Условие:

Определите **сумму номеров** всех процессов, которые **запустятся, но не успеют завершиться** за первые T = 41 мс.

Анализ:

- Процесс считается **запущенным**, если время начала < T.
- Процесс **не завершён** к T, если время завершения > T.
- Время начала = время завершения − время выполнения.

```python
# Читаем файл (тот же код)
times = {}
dependencies = {}

with open("22.txt") as f:
    for line in f:
        parts = line.strip().split("\t")
        pid = int(parts[0])
        time = int(parts[1])
        deps_str = parts[2]

        times[pid] = time
        if deps_str == "0":
            dependencies[pid] = []
        else:
            dependencies[pid] = [int(x) for x in deps_str.split(";")]

# Вычисляем время завершения (тот же код)
cache = {}

def completion_time(pid):
    if pid in cache:
        return cache[pid]

    deps = dependencies[pid]
    if not deps:
        cache[pid] = times[pid]
    else:
        max_dep = max(completion_time(dep) for dep in deps)
        cache[pid] = times[pid] + max_dep
    return cache[pid]

# Вычисляем время завершения для всех процессов
for pid in times:
    completion_time(pid)

T = 41
total = 0
for pid in times:
    finish = cache[pid]
    start = finish - times[pid]
    if start < T and finish > T:
        total += pid

print(total)
```

Логика:

- Процесс запущен к T, если `start < T`.
- Процесс не завершён к T, если `finish > T`.
- Суммируем ID таких процессов.

Сводка методов для №22

Задача	Метод	Ключевая формула
Время завершения	Рекурсия + кэш	`T[i] = time[i] + max(T[dep])`
Общее время	`max(T[i])`	Максимум по всем процессам
Процессы к моменту T	Сравнение `start` и `finish`	`start < T < finish`

Домашнее задание

Задание 1

Решите задачу 1 из открытого банка ФИПИ (№22).

Задание 2

Модифицируйте код, чтобы он выводил **время начала** каждого процесса.

Задание 3

Дана таблица процессов. Найдите количество процессов, которые **завершатся раньше 50 мс**.

Что мы сегодня изучили

Тема	Ключевые моменты
Модель процессов	ID, время, зависимости (граф)
Рекурсивная формула	`T[i] = time[i] + max(T[dep])`
Кэширование	Словарь `cache` для сохранения вычисленного времени
Задача с T	`start = finish - time`, проверка `start < T < finish`

> **Связь с ЕГЭ:** Задание №22 — 1 балл. Решается рекурсией с кэшем за 5 минут.

Чек-лист занятия

- [ ] Понимаю модель параллельных процессов
- [ ] Умею читать данные из файла в формате задачи
- [ ] Знаю рекурсивную формулу времени завершения
- [ ] Умею вычислять время начала и проверять условие по T
- [ ] Выполнено домашнее задание

Глава 5. Теория игр

Занятие 21. Теория игр: введение (задание №19)

План занятия

1. Модель игры: позиция, ходы, условие победы
2. Трёхчастный шаблон рекурсивной функции
3. Задание №19: Ваня выигрывает за 1 ход
4. Практика

> **Цель занятия:** Освоить базовый шаблон рекурсивной функции для теории игр и решать задание №19.

1. Модель игры: позиция, ходы, условие победы

Компоненты игры

1. **Начальная позиция** — количество камней в куче (или кучах).
2. **Ходы** — действия игрока: добавить, умножить, убрать.
3. **Условие победы** — достижение порога (≥ T или ≤ T).
4. **Победитель** — игрок, сделавший последний ход.

Что такое задание №19

Формулировка: Ваня выиграл своим первым ходом после чаще всего неудачного первого хода Пети.

Перевод:

- Петя не может выиграть сразу.
- Петя делает самый хороший для Вани ход, Ваня делает самый хороший для себя ход.

2. Шаблон рекурсивной функции

Структура шаблона

```
def game(position, turn):
    # Часть 1. Выход из рекурсии (база)
    if условие_победы(position):
        return turn == 0  # победа на последнем ходе игры
    if turn == 0:  # Кончились ходы
        return False

    # Часть 2. Список вариантов ходов
    moves = [
```

```
        game(position + 1, turn - 1),
        game(position * 2, turn - 1),
    ]

    # Часть 3. Возврат результата
    any(moves)  # Нужен хотя бы один победный ход
```

Разбор трёх частей

Часть 1 — База рекурсии:

- `turn` — сколько ходов осталось.
- Если позиция уже победная — выигрыш.
- Если ходы кончились (`turn == 0`) — и не достигнут выигрыш, то проигрыш.

Часть 2 — Варианты ходов:

- Создаём **список** результатов рекурсивных вызовов для всех возможных ходов.

Часть 3 — Возврат:

- `any(moves)` — если ходит Петя, ему нужен **хотя бы один** победный вариант.
- `all(moves)` — если ходит Ваня, Петя должен победить при **всех** вариантах.

3. Задание №19: Ваня выигрывает за 1 ход

Что это значит в шаблоне

Ваня выигрывает своим первым ходом → **turn = 2** (Петя — ход 2, Ваня — ход 1).

Петя не может выиграть сразу → из начальной позиции **нет** победного хода.

Упрощённый шаблон для №19

```
def game(s, turn):
    # Часть 1. База
    if s >= TARGET:
        return turn == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s + 1, turn - 1),
        game(s * 2, turn - 1),
    ]

    # Часть 3. Возврат
    return any(moves)

# Поиск S для №19
for S in range(1, TARGET):
    if game(S, 2):
```

```
        print(S)
        break
```

Условие `game(S, 2)`:

- `game(S, 2)` — Выигрыш в игре достигается на втором ходе игры. Должно быть `True`.

4. Практика

Задача 1 (две кучи)

Условие:

Две кучи камней. Ходы: +1 в одну из куч, или ×2 в одну из куч. Победа: сумма ≥ 231. Начало: первая куча = 17, вторая = S (1 ≤ S ≤ 213).

Вопрос: Минимальное S, при котором Ваня выигрывает своим первым ходом после неудачного хода Пети.

```
TARGET = 231
S1 = 17

def game(s1, s2, turn):
    # Часть 1. База
    if s1 + s2 >= TARGET:
        return turn == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов (все 4)
    moves = [
        game(s1 + 1, s2, turn - 1),
        game(s1, s2 + 1, turn - 1),
        game(s1 * 2, s2, turn - 1),
        game(s1, s2 * 2, turn - 1),
    ]

    # Часть 3. Возврат
    return any(moves)

for S in range(1, 214):
    if game(S1, S, 2):
        print(S)
        break
```

Задача 2 (одна куча, рост)

Условие:

Одна куча. Ходы: +1 или ×2. Победа: ≥ 65. Начало: S (1 ≤ S ≤ 64).

Вопрос: Минимальное S для №19.

```
TARGET = 65
```

```
def game(s, turn):
    # Часть 1. База
    if s ⩾ TARGET:
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s + 1, turn - 1),
        game(s * 2, turn - 1),
    ]

    # Часть 3. Возврат
    return any(moves)

for S in range(1, 65):
    if game(S, 2):
        print(S)
        break
```

Задача 3 (одна куча, уменьшение)

Условие:

Одна куча. Ходы: убрать 4, убрать 6, разделить на 2 (округление вниз). Победа: ≤ 1243 при **любом** ходе Пети. Начало: S > 1243.

Вопрос: Минимальное S для №19.

```
TARGET = 1243

def game(s, turn):
    # Часть 1. База (победа при уменьшении до ⩽ TARGET)
    if s ⩽ TARGET:
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s - 4, turn - 1),
        game(s - 6, turn - 1),
        game(s // 2, turn - 1),
    ]

    # Часть 3. Возврат
    if turn % 2 ≠ 0: # Блок меняется потому как Петя в этой задаче не
играет в поддавки
        return any(moves)
    else:
        return all(moves)

for S in range(TARGET + 1, 10000):
    if game(S, 2):
```

```
        print(S)
        break
```

Отличие от роста: в базе условие `s ⩽ TARGET` вместо `s ⩾ TARGET`.

Шаблон для всех задач №19

```
TARGET = ...  # Порог победы

def game(позиция, turn):
    # Часть 1. База
    if условие_победы(позиция):
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(ход1, turn - 1),
        game(ход2, turn - 1),
        ...
    ]

    # Часть 3. Возврат
    if turn % 2 != 0:
        return any(moves)
    else:
        return all(moves)

# Поиск ответа
for S in диапазон:
    if game(S, 2):
        print(S)
        break
```

Компонент	Что делает
`turn % 2 == 0`	Ход Пети → `all(moves)`
`turn % 2 == 1`	Ход Вани → `any(moves)`
`game(S, 2)`	Ваня выигрывает за 2 хода

Домашнее задание

Задание 1

Одна куча. Ходы: +2, ×3. Победа: ≥ 100. Найдите минимальное S для №19.

Задание 2

Одна куча. Ходы: −3, −5, //2. Победа: ≤ 10. Найдите минимальное S > 10 для №19.

Задание 3

Две кучи. Ходы: +2 в одну, ×2 в одну. Победа: сумма ≥ 100. Первая куча = 10. Найдите минимальное S для №19.

Что мы сегодня изучили

Тема	Ключевые моменты
Трёхчастный шаблон	База → варианты → возврат
`any(moves)`	Хотя бы один победный ход
`all(moves)`	Все ходы должны вести к победе
№19	`game(S, 2)`
Рост/уменьшение	`⩾ TARGET` или `⩽ TARGET` в базе

> **Связь с ЕГЭ:** Этот шаблон — основа для всех задач №19–21. Запомните его структуру.

На следующем занятии (Занятие 22)

Тема: Теория игр: any/all (задание №20)

- Задачи на 2 хода
- Усложнение логики `any` / `all`
- Поиск минимального/максимального S

Чек-лист занятия

- ☐ Знаю трёхчастный шаблон рекурсивной функции
- ☐ Понимаю `any` и `all`
- ☐ Умею решать №19 через `game(S,2)`
- ☐ Различаю рост (≥) и уменьшение (≤) в базе
- ☐ Выполнено домашнее задание

Занятие 22. Теория игр: any/all (задание №20)

План занятия

1. Повторение трёхчастного шаблона
2. `any()` — ходящий ищет хотя бы один победный ход

3. `all()` — противник мешает
4. Задание №20: Петя выигрывает вторым ходом
5. Практика

> **Цель занятия:** Понять логику `any` / `all` и научиться решать задание №20.

1. Повторение трёхчастного шаблона

Структура (из занятия 21)

```
TARGET = ...

def game(позиция, turn):
    # Часть 1. База
    if условие_победы(позиция):
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(ход1, turn - 1),
        game(ход2, turn - 1),
        ...
    ]

    # Часть 3. Возврат
    if turn % 2 != 0:
        return any(moves)
    else:
        return all(moves)
```

Отличие №20 от №19

Задание	Условие поиска	Смысл
№19	`game(S, 2)`	Ваня выигрывает своим первым ходом
№20	`not game(S, 1) and game(S, 3)`	Петя выигрывает своим вторым ходом

№20: Петя не может выиграть сразу, но выигрывает на **третьем ходе** (своём втором).

```
Ход 1: Петя  (turn=3)
Ход 2: Ваня  (turn=2)
Ход 3: Петя  (turn=1) — победа
```

2. `any()` — ходящий игрок ищет хотя бы один победный ход

Логика

Когда ходит **Петя** (turn нечётный), он выбирает ход. Ему нужен **хотя бы один** вариант, ведущий к победе.

```python
if turn % 2 != 0:  # Ход Пети
    return any(moves)  # True, если есть победный ход
```

Дерево решений

```
Петя (turn=3): нужно any
  ├─ Ход +1: Ваня(turn=2) → ...
  └─ Ход ×2: Ваня(turn=2) → ...
```

Петя победит, если **хотя бы один** из этих вариантов приведёт к победе.

3. `all()` — противник мешает

Логика

Когда ходит **Ваня** (turn чётный), он будет **мешать** Пете. Для Петиной победы нужно, чтобы **все** возможные ходы Вани вели к победе Пети.

```python
if turn % 2 == 0:  # Ход Вани
    return all(moves)  # True, только если ВСЕ ходы ведут к победе Пети
```

Дерево решений

```
Ваня (turn=2): нужно all
  ├─ Ход +1: Петя(turn=1) → должен победить
  └─ Ход ×2: Петя(turn=1) → должен победить
```

Если Ваня может сделать ход, после которого Петя **не** победит — Ваня сделает именно его. Поэтому для победы Пети нужно, чтобы из **всех** возможных позиций после хода Вани Петя мог выиграть.

4. Задание №20: Петя выигрывает вторым ходом

Условие

> Петя не может выиграть за один ход, но может выиграть своим вторым ходом независимо от хода Вани.

Перевод на шаблон:

```python
not game(S, 1) and game(S, 3)
```

- `game(S, 1)` — Петя выигрывает за 1 ход? Должно быть `False`.
- `game(S, 3)` — Петя выигрывает за 3 хода (свой второй)? Должно быть `True`.

Ходы по уровням

turn	Кто ходит	`turn % 2 ≠ 0`	Логика
3	Петя	Да (`any`)	Ищет победный ход
2	Ваня	Нет (`all`)	Все ходы ведут к победе Пети
1	Петя	Да (`any`)	Должен победить немедленно
0	—	—	Ходы кончились → `False`

Поиск нескольких значений S

В №20 часто просят **два наименьших** значения S.

```
answers = []
for S in диапазон:
    if not game(S, 1) and game(S, 3):
        answers.append(S)
        if len(answers) == 2:
            break
print(''.join(str(x) for x in answers))
```

5. Практика

Задача 1 (две кучи)

Условие:

Две кучи. Ходы: +1 в одну или ×2 в одну. Победа: сумма ≥ 231. Первая куча = 17, вторая = S (1 ≤ S ≤ 213).

Вопрос: Два наименьших S, при которых Петя не может выиграть за 1 ход, но выигрывает вторым ходом.

```
TARGET = 231
S1 = 17

def game(s1, s2, turn):
    # Часть 1. База
    if s1 + s2 ≥ TARGET:
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s1 + 1, s2, turn - 1),
        game(s1, s2 + 1, turn - 1),
        game(s1 * 2, s2, turn - 1),
        game(s1, s2 * 2, turn - 1),
    ]

    # Часть 3. Возврат
    if turn % 2 ≠ 0:
```

```
        return any(moves)
    else:
        return all(moves)

answers = []
for S in range(1, 214):
    if not game(S1, S, 1) and game(S1, S, 3):
        answers.append(S)
        if len(answers) == 2:
            break

print(''.join(str(x) for x in answers))
```

Задача 2 (одна куча, рост)

Условие:

Одна куча. Ходы: +1 или ×2. Победа: ≥ 65. Начало: S (1 ≤ S ≤ 64).

Вопрос: Два наименьших S для №20.

```
TARGET = 65

def game(s, turn):
    # Часть 1. База
    if s >= TARGET:
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s + 1, turn - 1),
        game(s * 2, turn - 1),
    ]

    # Часть 3. Возврат
    if turn % 2 != 0:
        return any(moves)
    else:
        return all(moves)

answers = []
for S in range(1, 65):
    if not game(S, 1) and game(S, 3):
        answers.append(S)
        if len(answers) == 2:
            break

print(''.join(str(x) for x in answers))
```

Задача 3 (одна куча, уменьшение)

Условие:

Одна куча. Ходы: убрать 4, убрать 6, разделить на 2 (округление вниз). Победа: ≤ 1243. Начало: S > 1243.

Вопрос: Два наименьших S для №20.

```
TARGET = 1243

def game(s, turn):
    # Часть 1. База (победа при уменьшении)
    if s ⩽ TARGET:
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s - 4, turn - 1),
        game(s - 6, turn - 1),
        game(s // 2, turn - 1),
    ]

    # Часть 3. Возврат
    if turn % 2 ≠ 0:
        return any(moves)
    else:
        return all(moves)

answers = []
for S in range(TARGET + 1, 10000):
    if not game(S, 1) and game(S, 3):
        answers.append(S)
        if len(answers) == 2:
            break

print(''.join(str(x) for x in answers))
```

Сводка: turn, any/all и задания

turn	Ход	Логика	Кто ходит
1	Первый	`any`	Петя
2	Второй	`all`	Ваня
3	Третий	`any`	Петя
4	Четвёртый	`all`	Ваня

Задание	Условие	Смысл
№19	`not game(S, 1) and game(S, 2)`	Ваня выигрывает первым ходом
№20	`not game(S, 1) and game(S, 3)`	Петя выигрывает вторым ходом
№21	`game(S, 3) and not game(S, 1)`	Разные условия

Домашнее задание

Задание 1

Одна куча. Ходы: +2, ×3. Победа: ≥ 100. Найдите два наименьших S для №20.

Задание 2

Одна куча. Ходы: −3, −5, //2. Победа: ≤ 10. Найдите два наименьших S > 10 для №20.

Задание 3

Две кучи. Ходы: +2 в одну, ×2 в одну. Победа: сумма ≥ 100. Первая куча = 10. Найдите два наименьших S для №20.

Что мы сегодня изучили

Тема	Ключевые моменты
`any(moves)`	Хотя бы один победный ход (Петя)
`all(moves)`	Все ходы ведут к победе (Ваня)
Переключение	`turn % 2 == 0` → `any`, иначе `all`
№20	`not game(S, 1) and game(S, 3)`
Два значения	Собираем в список, останавливаемся после двух

> **Связь с ЕГЭ:** Задание №20 — 1 балл. Тот же шаблон, что и №19, только `turn=3`.

На следующем занятии (Занятие 23)

Тема: Теория игр: полный анализ (задание №21)

- Полная рекурсия до конца игры
- Перебор всех начальных S
- Поиск минимального/максимального S с выигрышной стратегией

Чек-лист занятия

- ☐ Понимаю разницу между `any` и `all`
- ☐ Знаю, когда `turn % 2 == 0` (any), а когда `turn % 2 == 0` (all)
- ☐ Умею решать №20 через `not game(S, 1) and game(S, 3)`
- ☐ Умею находить несколько значений S
- ☐ Выполнено домашнее задание

Занятие 23. Теория игр: полный анализ (задание №21)

План занятия

1. Повторение трёхчастного шаблона
2. Задание №21: оба условия одновременно
3. Как считать: `game(S, 2)` и `game(S, 4)`
4. Практика

> **Цель занятия:** Научиться решать задание №21 — полный анализ игры с двумя условиями.

1. Повторение трёхчастного шаблона

```
TARGET = ...

def game(позиция, turn):
    # Часть 1. База
    if условие_победы(позиция):
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(ход1, turn - 1),
        game(ход2, turn - 1),
        ...
    ]

    # Часть 3. Возврат
    if turn % 2 != 0:
        return any(moves)
    else:
        return all(moves)
```

2. Задание №21: оба условия одновременно

Типовая формулировка

> Найдите минимальное S, при котором одновременно:
>
> 1. У Вани есть выигрышная стратегия, позволяющая выиграть **первым или вторым** ходом.
> 2. У Вани **нет** стратегии, гарантирующей выигрыш **первым** ходом.

Перевод на язык шаблона

Условие 1: Ваня может выиграть первым или вторым ходом.

- Ваня выигрывает своим первым ходом → `game(S, 2)`.
- Ваня выигрывает своим первым или вторым ходом → `game(S, 4)`.

Условие 2: Ваня НЕ может гарантированно выиграть первым ходом.

- `not game(S, 2)`.

Итоговое условие:

`not game(S, 2) and game(S, 4)`.

3. Как считать: `game(S, 2)` и `game(S, 4)`

`game(S, 2)` — Ваня выигрывает первым ходом

Из начальной позиции Петя делает любой ход (не победный). Из полученной позиции Ваня должен иметь победный ход.

```
game(S, 2)  # True, если Ваня выигрывает на ходе 2
```

`game(S, 4)` — Ваня выигрывает первым или вторым ходом

Петя → Ваня (возможно победный) → Петя (не победный) → Ваня (победный).

```
game(S, 4)  # True, если Ваня выигрывает либо на на ходе 4 либо на ходе 2
```

Сводка условий

Зада-ние	Условие	Смысл
№19	`game(S, 2)`	Ваня выигрывает первым ходом
№20	`not game(S, 1) and game(S, 3)`	Петя выигрывает вторым ходом
№21	`not game(S, 2) and game(S, 4)`	Ваня выигрывает первым или вторым ходом но не строго первым

4. Практика

Задача 1 (две кучи)

Условие:

Две кучи. Ходы: +1 в одну или ×2 в одну. Победа: сумма ≥ 231. Первая куча = 17, вторая = S ($1 \le S \le 213$).

Вопрос: Минимальное S, при котором у Вани есть стратегия выиграть первым или вторым ходом, но нет стратегии выиграть первым ходом.

```
TARGET = 231
S1 = 17

def game(s1, s2, turn):
    # Часть 1. База
    if s1 + s2 ⩾ TARGET:
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s1 + 1, s2, turn - 1),
        game(s1, s2 + 1, turn - 1),
        game(s1 * 2, s2, turn - 1),
        game(s1, s2 * 2, turn - 1),
    ]

    # Часть 3. Возврат
    if turn % 2 ≠ 0:
        return any(moves)
    else:
        return all(moves)

# Ищем минимальное S
for S in range(1, 214):
    if not game(S1, S, 2) and game(S1, S, 4):
        print(S)
        break
```

Задача 2 (одна куча)

Условие:

Одна куча. Ходы: +1 или ×2. Победа: ≥ 65. Начало: S (1 ≤ S ≤ 64).

Вопрос: Минимальное S для №21.

```
TARGET = 65

def game(s, turn):
    # Часть 1. База
    if s ⩾ TARGET:
        return turn % 2 == 0
    if turn == 0:
        return False

    # Часть 2. Варианты ходов
    moves = [
        game(s + 1, turn - 1),
        game(s * 2, turn - 1),
    ]
```

```
    # Часть 3. Возврат
    if turn % 2 != 0:
        return any(moves)
    else:
        return all(moves)

# Ищем минимальное S
for S in range(1, 65):
    if not game(S, 2) and game(S, 4):
        print(S)
        break
```

Полная сводка условий для №19–21

Задание	Условие	Кто выигрывает	На каком ходу игрока (общий ход в игре)
№19	`game(S, 2)`	Ваня	Первом (ход 2)
№20	`not game(S, 1) and game(S, 3)`	Петя	Втором (ход 3)
№21	`not game(S, 2) and game(S, 4)`	Ваня	Втором или первом (ход 4 или 2 но не строго 2)

Как запомнить

- **№19:** Ваня первым ходом → `game(S, 2)`.
- **№20:** Петя вторым ходом → `game(S, 3)`.
- **№21:** Ваня вторым ходом → `game(S, 4)`.

Домашнее задание

Задание 1

Одна куча. Ходы: +2, ×3. Победа: ≥ 100. Найдите минимальное S для №21.

Задание 2

Одна куча. Ходы: −3, −5, //2. Победа: ≤ 10. Найдите минимальное S > 10 для №21.

Задание 3

Две кучи. Ходы: +2 в одну, ×2 в одну. Победа: сумма ≥ 100. Первая куча = 10. Найдите минимальное S для №21.

Что мы сегодня изучили

Тема	Ключевые моменты
№21 — два условия	«Есть стратегия» и «нет стратегии»
`game(S, 4)`	Ваня выигрывает вторым ходом
Условие №21	`not game(S, 2) and game(S, 4)`
Сводка №19–21	Разные `turn`: 2, 3, 4

> **Связь с ЕГЭ:** Задание №21 — 1 балл. Тот же шаблон, только `turn=4`.

Чек-лист занятия

- ☐ Знаю трёхчастный шаблон для теории игр
- ☐ Понимаю смысл `game(S, 2)` и `game(S, 4)`
- ☐ Умею решать №21 через `not game(S, 2) and game(S, 4)`
- ☐ Знаю сводку условий для №19–21
- ☐ Выполнено домашнее задание

Глава 6. Динамическое программирование

Занятие 24. ДП: нисходящее vs восходящее (задание №16, сложное)

План занятия

1. Что такое динамическое программирование (определение Беллмана)
2. Два признака ДП: оптимальная подструктура и перекрывающиеся подзадачи
3. Два способа реализации: мемоизация (нисходящее) и табуляция (восходящее)
4. Когда что выбирать: глубина рекурсии и объём кэша
5. Цифровое ДП (digit DP) для задач с ограничением на число
6. Практика

> **Цель занятия:** Понять суть динамического программирования, научиться выбирать между нисходящим и восходящим подходом и освоить технику цифрового ДП для сложных задач №16.

1. Что такое динамическое программирование

Классическое определение (Ричард Беллман, 1950-е)

Динамическое программирование — это метод решения сложных задач путём разбиения их на **перекрывающиеся подзадачи**, решения каждой подзадачи **один раз** и сохранения результата для повторного использования.

Два обязательных признака ДП

1. **Оптимальная подструктура**
   Оптимальное решение всей задачи содержит оптимальные решения её подзадач.

2. **Перекрывающиеся подзадачи**
   Одни и те же подзадачи возникают многократно при рекурсивном решении. Если их не сохранять, они будут вычисляться снова и снова.

Простой пример: числа Фибоначчи

`F(n) = F(n-1) + F(n-2)` — оптимальная подструктура есть.
`F(5)` вызывает `F(4)` и `F(3)`, `F(4)` вызывает `F(3)` и `F(2)` — подзадачи перекрываются.

2. Рекурсия ≠ ДП. Когда рекурсия становится ДП

Три уровня

1. **Просто рекурсия (без памяти)**

   Вычисляет одни и те же подзадачи многократно — экспоненциальный взрыв. Это **не ДП**.

```python
def fib(n):
    if n <= 1: return n
    return fib(n-1) + fib(n-2)
```

2. **Рекурсия + кэш = нисходящее ДП (мемоизация)**

   Подзадачи решаются один раз, результаты сохраняются. Это уже **ДП**.

```python
@lru_cache(maxsize=None)
def fib(n):
    if n <= 1: return n
    return fib(n-1) + fib(n-2)
```

3. **Цикл + массив = восходящее ДП (табуляция)**

   Без рекурсии, заполняем таблицу от меньших значений к большим. Тоже **ДП**.

```python
def fib_table(n):
    dp = [0]*(n+1); dp[1] = 1
    for i in range(2,n+1): dp[i]=dp[i-1]+dp[i-2]
    return dp[n]
```

4. **Восходящее ДП без таблицы**

```python
def fib_table(n):
    dp_2 = 0; dp_1 = 1
    for i in range(2,n+1):
        dp=dp_1+dp_2
        dp_2=dp_1
        dp_1=dp
    return dp
```

Вывод: рекурсия с сохранением результатов — это ДП. Рекурсия без сохранения — нет.

3. Способы реализации ДП

Подход	Реализация	Название	Плюсы	Минусы
Сверху вниз	Рекурсия + кэш	Мемоиза-ция	Легко писать, порядок вычислений автоматический	Глубина рекурсии ограничена, кэш может расти
Снизу вверх	Цикл + мас-сив	Табуляция	...	Нужно продумать порядок обхода состояний
Снизу вверх	Цикл без массива	Итерация	Память не растет	Нужно продумать порядок обхода состояний

В нашем курсе

- Занятие 17: мемоизация (`@lru_cache`) для задач №16, 23.

- Занятие 21-22: рекурсивные функции `game()` с кэшем — это тоже ДП.
- Это занятие: расширяем арсенал восходящим ДП и цифровым ДП.

4. Когда что выбирать

Ситуация	Подход
Глубина рекурсии ≤ 1000, состояний немного	Нисходящее (`@lru_cache`)
Глубина > 1000 (например, n до 10^6)	Восходящее
Кэш может разрастись до миллионов записей	Восходящее
Сложные переходы, нелинейный порядок	Нисходящее (проще написать)

Для задания №16

Обычные задачи (F(n) через F(n-1) или F(n//2)) отлично решаются нисходящим ДП с кэшем. Но есть особый класс задач: **подсчёт количества чисел ≤ N с заданным свойством F(n)**. Для них применяют **цифровое ДП** (digit DP) — мощную технику нисходящего ДП с мемоизацией.

5. Цифровое ДП (digit DP) для задач с ограничением на число

Когда применяется

Нужно посчитать количество чисел n ≤ MAX, для которых задана некоторая функция от цифр (сумма, произведение и т.п.).

Пример: Сколько чисел n < 2^{63} имеют сумму цифр = 159?

Перебрать все числа до $9.2 \cdot 10^{18}$ невозможно. Но можно перебирать **цифры** числа, начиная со старшей, и считать сумму.

Параметры состояния

- `pos` — сколько цифр осталось дописать (или текущая позиция слева).
- `sum_digits` — накопленная сумма (или другая характеристика).
- `tight` — флаг, указывающий, совпадает ли текущий префикс с верхней границей.

Флаг `tight`:

- Если `tight = True`, то следующая цифра не может превышать соответствующую цифру MAX.
- Если `tight = False`, то все цифры от 0 до 9 разрешены.

Шаблон цифрового ДП (нисходящее)

```
@lru_cache(maxsize=None)
def dp(pos, sum_digits, tight):
    if pos == len(digits):  # Все цифры обработаны
        return 1 if sum_digits == target else 0
```

```
    limit = digits[pos] if tight else 9
    total = 0
    for d in range(limit + 1):
        total += dp(pos + 1, sum_digits + d, tight and (d == limit))
    return total
```

Запуск: `dp(0, 0, True)` — начинаем со старшей цифры, сумма 0, флаг tight = True.

Почему это работает быстро

Состояний: `len(digits) × (max_sum) × 2`. Для чисел до 2^{63} это примерно 19 × (9×19) × 2 ≈ 6500 — мгновенно.

6. Практика

Задача 1 (сумма цифр = 159)

Условие:

F(n) = n, если n < 10; F(n) = F(n % 10) + F(n // 10), если n ≥ 10.

Фактически F(n) — это сумма цифр числа n. Найти количество n < 2^{63}, для которых F(n) = 159.

```
from functools import lru_cache

MAX = 2**63 - 1
digits = list(map(int, str(MAX)))
target = 159

@lru_cache(maxsize=None)
def dp(pos, s, tight):
    if pos == len(digits):
        return 1 if s == target else 0
    if s > target:
        return 0

    limit = digits[pos] if tight else 9
    total = 0
    for d in range(limit + 1):
        total += dp(pos + 1, s + d, tight and (d == limit))
    return total

print(dp(0, 0, True))
```

Задача 2 (произведение «цифр» в 15-ричной системе)

Условие:

F(n) = n, если n < 15; F(n) = F(n % 15) × F(n // 15), если n ≥ 15.

F(n) — произведение цифр 15-ричной записи числа n. Найти количество n ≤ 3^{40}, для которых F(n) = 7560.

```python
from functools import lru_cache

MAX = 3**40
# Перевод MAX в 15-ричную систему
digits = []
temp = MAX
while temp > 0:
    digits.append(temp % 15)
    temp //= 15
digits = digits[::-1]

target = 7560

@lru_cache(maxsize=None)
def dp(pos, prod, tight, started):
    if pos == len(digits):
        return 1 if (prod == target and started) else 0
    if prod > target:
        return 0

    limit = digits[pos] if tight else 14
    total = 0
    for d in range(limit + 1):
        if not started and d == 0:
            total += dp(pos + 1, prod, tight and (d == limit), False)
        else:
            new_prod = prod * d if started else d
            total += dp(pos + 1, new_prod, tight and (d == limit), True)
    return total

print(dp(0, 1, True, False))
```

Итоговая сводка

Подход	Когда применять	Примеры в курсе
Нисходящее ДП (мемоизация)	Глубина ≤ 1000, состояний немного	№16, 19-21, 23, digit DP
Восходящее ДП (табуляция)	Глубина > 1000, нужна экономия памяти	№16 (глубокие), №27

Домашнее задание

Задание 1

Сколько натуральных чисел $n \le 10^9$ имеют сумму цифр, равную 42? Решите цифровым ДП.

Задание 2

$F(n) = n$ при $n < 5$; $F(n) = F(n // 5) + F(n \% 5)$ при $n \ge 5$. Найдите количество $n \le 5^{10}$, для которых $F(n) = 30$.

Задание 3

Перепишите функцию `fib` восходящим ДП и сравните скорость для n = 500.

Что мы сегодня изучили

Тема	Ключевые моменты
Определение ДП	Метод решения с перекрывающимися подзадачами и сохранением результатов
Два признака ДП	Оптимальная подструктура + перекрывающиеся подзадачи
Рекурсия ≠ ДП	Только с сохранением (мемоизацией) рекурсия становится ДП
Нисходящее vs восходящее	Плюсы и минусы, когда что выбирать
Цифровое ДП	`pos, sum/prod, tight, started` для подсчёта чисел ≤ MAX

> **Связь с ЕГЭ:** Мы уже использовали ДП (мемоизацию) в №16, 19-21, 23. Теперь расширяем инструментарий восходящим ДП и цифровым ДП для самых сложных задач.

На следующем занятии (Занятие 25)

Тема: ДП по длине слова (задание №8, сложное)

- Когда брутфорс не справляется
- Состояния: `dp[длина][последний_символ]`
- Подсчёт слов с ограничениями

Чек-лист занятия

- ☐ Знаю классическое определение ДП (Беллман)
- ☐ Понимаю разницу между просто рекурсией и рекурсией с кэшем
- ☐ Могу выбрать между нисходящим и восходящим подходом
- ☐ Знаю шаблон цифрового ДП с флагом `tight`
- ☐ Умею учитывать ведущие нули (флаг `started`)
- ☐ Выполнено домашнее задание

Занятие 25. ДП по длине слова (задание №8, сложное)

План занятия

1. Когда брутфорс не справляется: длина > 8, алфавит > 10
2. ДП-таблица: `dp[длина][состояние]`
3. Состояние: последний символ, счётчики, маски
4. Переходы между состояниями

5. Практика

> **Цель занятия:** Освоить динамическое программирование для подсчёта объектов с ограничениями, когда полный перебор невозможен. Это ключ к сложным задачам №8.

1. Когда брутфорс не справляется

На прошлых занятиях мы перебирали слова длиной до 8, используя `itertools.product`. Но если длина 12, а алфавит 9, количество вариантов $9^{12} \approx 2.8 \cdot 10^{11}$ — перебор займёт часы. А если нужно ещё и ограничение на число (например, $\leq 855\,000\,000$), брутфорс тем более не подходит.

Решение: считать количество объектов **по частям**, постепенно наращивая длину и сохраняя только необходимую информацию о текущем префиксе. Это — **динамическое программирование по длине**.

2. ДП-таблица: `dp[длина][состояние]`

Основная идея

Пусть мы строим слово (или число) посимвольно. Состояние описывает всё, что нужно знать о текущем префиксе для проверки ограничений на следующие символы.

Типичные состояния:

- последний поставленный символ (для проверки соседей),
- количество определённых символов (например, число использованных цифр),
- маска использованных цифр (для учёта уникальности).

Мы вычисляем `dp[pos][state]` — количество способов дописать слово от позиции `pos` до конца, если текущее состояние `state`. Обычно используют рекурсию с мемоизацией (нисходящее ДП).

Шаблон (рекурсивный)

```
@lru_cache(maxsize=None)
def dp(pos, last_digit, ...):
    if pos == L:          # Достигли нужной длины
        return 1           # или проверяем финальное условие
    total = 0
    for d in range(base): # перебираем следующий символ
        if условие_допустимости(last_digit, d, ...):
            total += dp(pos+1, d, ...)
    return total
```

Для больших длин (L > 1000) рекурсия может быть глубокой — тогда переписывают на восходящее ДП с циклами.

3. Состояние: последний символ, счётчики, маски

Только ограничения на соседей

Состояние: `(pos, last)`. Подходит для задач «никакие две одинаковые цифры не стоят рядом», «чётная/нечётная чередуются».

Ограничения на количество символов

Состояние: `(pos, last, count_A)`. Например, «ровно две буквы A».

Ограничения на уникальность (различные цифры)

Состояние: `(pos, used_mask)`, где `used_mask` — битовая маска длиной base (до 15–16 бит). Для основания 15 маска помещается в целое число (15 бит).

Ограничение на числовое значение (≤ MAX)

Добавляется флаг `tight` (как в цифровом ДП), означающий, что текущий префикс совпадает с MAX.

4. Переходы между состояниями

Для каждого состояния перебираем все возможные следующие символы `d`. Если пара `(last, d)` удовлетворяет условию, вызываем рекурсию с обновлёнными параметрами:

- `pos + 1`,
- новый `last = d`,
- новый `used_mask | (1 << d)`,
- новый `tight = tight and (d == limit)`.

Количество состояний обычно невелико: `длина × base × (размер маски или счётчика) × 2`. Для base=15, L=8 (или 9) это тысячи состояний — мгновенно.

5. Практика

Задача 1 (15-ричные числа ≤ 855 млн, ровно 8 различных цифр)

Условие: Найти количество натуральных чисел, не превышающих 855 000 000, запись которых в системе счисления с основанием 15 содержит ровно 8 различных цифр.

Анализ:

- Максимальное число: $N = 855\,000\,000$.
- Переводим его в 15-ричную систему. Длина записи — до 8 цифр (проверим: $15^7 = 170\,859\,375$, $15^8 = 2\,562\,890\,625$, значит, 855 млн — 8-значное в 15-ричной).

- Число может быть короче (ведущие нули не пишутся), но для подсчёта «ровно 8 различных цифр» длина должна быть ≥ 8. Так как 855 млн — 8-значное, все числа с 8 различными цифрами будут 8-значными (или 9-значными, но они превысят лимит).
- Задача: подсчитать 8-значные 15-ричные числа (первая цифра не 0) ≤ N, содержащие ровно 8 различных цифр.

Состояния: `pos`, `mask` (15 бит), `tight`, `started`.

```
from functools import lru_cache

N = 855_000_000
# перевод в 15-ричную
digits = []
temp = N
while temp:
    digits.append(temp % 15)
    temp //= 15
digits = digits[::-1]  # старшие впереди
L = len(digits)  # 8

@lru_cache(maxsize=None)
def dp(pos, mask, tight, started):
    if pos == L:
        # считаем количество установленных битов в mask
        return 1 if started and bin(mask).count('1') == 8 else 0
    limit = digits[pos] if tight else 14
    total = 0
    for d in range(limit + 1):
        if not started and d == 0:
            # ведущий ноль — число ещё не началось, маска пустая
            total += dp(pos+1, 0, tight and (d == limit), False)
        else:
            total += dp(pos+1, mask | (1 << d), tight and (d == limit), Tr
ue)
    return total

print(dp(0, 0, True, False))
```

Ответ: подставьте, программа вычислит.

Задача 2 (подходящие 12-значные 9-ричные числа)

Условие: Ряд из двух цифр подходящий, если:

1. сумма чётна и вторая цифра больше первой;
2. сумма нечётна и вторая цифра меньше первой. Число подходящее, если все соседние пары подходящие. Найти количество подходящих 12-значных 9-ричных чисел (первая цифра не 0).

Анализ:

- Алфавит 0..8, первая цифра 1..8.
- Условие на соседей зависит только от двух цифр. Состояние: `(pos, last)`.
- База: `pos == 12` → return 1.

```python
from functools import lru_cache

L = 12
base = 9

def ok(a, b):
    s = a + b
    if s % 2 == 0:
        return b > a
    else:
        return b < a

@lru_cache(maxsize=None)
def dp(pos, last):
    if pos == L:
        return 1
    total = 0
    for d in range(base):
        if pos == 0 and d == 0:  # первая цифра не 0
            continue
        if pos == 0 or ok(last, d):
            total += dp(pos+1, d)
    return total

print(dp(0, -1))
```

Пояснение: при `pos == 0` условие `ok` не проверяем, просто ставим первую цифру. `-1` как `last` — любое, условие не проверяется.

Сводка методов для сложного №8

Тип задачи	Состояния	Пример
Ограничения на соседей	`(pos, last)`	Подходящие числа
Ограничение на количество цифр	`(pos, last, count)`	Ровно k цифр X
Уникальные цифры	`(pos, mask)`	Числа с разными цифрами
+ ограничение на значение	`(pos, mask, tight, started)`	Числа ≤ N

Все эти методы — частные случаи **динамического программирования по длине (цифрового ДП)**.

Домашнее задание

Задание 1

Сколько 10-значных 8-ричных чисел, в которых никакие две чётные цифры не стоят рядом?

Задание 2

Сколько натуральных чисел, не превышающих 10^9, в 12-ричной записи которых ровно 5 различных цифр?

Задание 3

Выведите ответ для задачи 2 из практики (подходящие 12-значные 9-ричные числа) и проверьте его для малых длин (L=2,3) с помощью `itertools.product`.

Что мы сегодня изучили

Тема	Ключевые моменты
ДП по длине слова	Состояние `(позиция, последний символ, ...)`
Цифровое ДП с маской	`mask` для учёта использованных цифр
Флаг `tight` и `started`	Для ограничения на значение числа
Переходы	Перебор следующего символа, проверка условий

> **Связь с ЕГЭ:** Эти методы решают любую сложную задачу №8 и некоторые задачи №16. Это вершина комбинаторного ДП в школьной информатике.

На следующем занятии (Занятие 26)

Тема: ДП в подсчёте траекторий (задание №23, сложное)

- Траектории с обязательными и запрещёнными числами
- ДП снизу вверх: заполнение массива

Чек-лист занятия

- ☐ Умею определять, когда нужен ДП, а не брутфорс
- ☐ Могу построить состояние для задачи с ограничениями на соседей
- ☐ Знаю, как использовать битовую маску для уникальных цифр
- ☐ Понимаю флаги `tight` и `started`
- ☐ Выполнено домашнее задание

Занятие 26. ДП в подсчёте траекторий (задание №23, сложное)

План занятия

1. Повторение: базовый подсчёт траекторий
2. Траектория содержит число X: разбиение на два этапа
3. Траектория не содержит число Y: параметр-флаг

4. Комбинация условий: содержит И не содержит
5. Когда рекурсии не хватает: восходящее ДП
6. Практика

> **Цель занятия:** Научиться решать усложнённые задачи №23 с комбинацией условий на траекторию.

1. Повторение: базовый подсчёт траекторий

Шаблон

```python
from functools import lru_cache

@lru_cache(maxsize=None)
def paths(start, end):
    if start == end:
        return 1
    if start > end:
        return 0
    return sum(paths(start + step, end) for step in moves)
```

Где `moves` — список допустимых ходов (например, `[1, 2, 3]`).

Что мы умеем

- Считать количество программ из A в B.
- Добавлять условие «содержит X» (разбиением на два этапа).
- Добавлять условие «не содержит Y» (флагом `forbidden`).

Сегодня мы соединим оба условия и научимся решать задачи, где рекурсия с флагами становится громоздкой — с помощью восходящего ДП.

2. Траектория содержит число X

Разбиение на два этапа

Все траектории из A в B, проходящие через X, разбиваются на две независимые части:

1. Из A в X.
2. Из X в B.

Общее количество = `paths(A, X) * paths(X, B)`.

```python
result = paths(1, 7) * paths(7, 35)
```

Почему умножение: каждый путь из A в X можно скомбинировать с любым путём из X в B. Комбинации независимы.

Ограничение

Это работает, только если **нет других условий** (запретов). Если есть запрет на число Y, нужно, чтобы обе части траектории его обходили.

3. Траектория не содержит число Y

Флаг `forbidden` в рекурсии

```
@lru_cache(maxsize=None)
def paths(start, end, forbidden):
    if start == end:
        return 1
    if start > end or start == forbidden:
        return 0
    return sum(paths(start + step, end, forbidden) for step in moves)
```

Недостаток: функция зависит от трёх параметров, кэш хранит результаты для каждой тройки `(start, forbidden)` — состояний $O(N^2)$, если forbidden меняется.

4. Комбинация условий: содержит И не содержит

Задача

Траектория должна содержать X и не содержать Y.

Решение через разбиение

Разбиваем на два этапа, на каждом этапе запрещаем Y:

```
result = paths(A, X, forbidden=Y) * paths(X, B, forbidden=Y)
```

Важно: число X не должно совпадать с Y (иначе условий не выполнить). X само по себе не запрещено, поэтому в первом этапе конечная точка X разрешена (мы не проверяем `start == forbidden` для конечной точки, только для промежуточных). В функции `paths` выше условие `start == forbidden` срабатывает на **промежуточных** шагах, а конечная точка проверяется отдельно (`start == end`). Так что X может совпадать с конечной точкой.

Если запретов несколько

```
forbidden_set = {Y1, Y2, Y3}

@lru_cache(maxsize=None)
def paths(start, end):
    if start == end:
        return 1
    if start > end or start in forbidden_set:
        return 0
    return sum(paths(start + step, end) for step in moves)
```

Кортеж запретов фиксирован, поэтому он не влияет на размер кэша.

5. Когда рекурсии не хватает: восходящее ДП

Проблема рекурсивного подхода

Если нужно найти **количество траекторий для всех начальных значений** или если параметров слишком много (например, траектория должна содержать X, не содержать Y, и при этом иметь определённую сумму чисел), рекурсия с флагами становится громоздкой.

Восходящее ДП (итеративное)

Заводим массив `dp[i]` — количество программ из A в i. Заполняем от A до B:

```
dp = [0] * (end + 1)
dp[A] = 1
for i in range(A, end + 1):
    if i in forbidden_set:
        continue
    for step in moves:
        if i + step <= end:
            dp[i + step] += dp[i]
```

Плюсы:

- Нет ограничения на глубину рекурсии.
- Можно легко добавить несколько массивов для разных условий.
- Работает быстро для end до 10^6.

Два массива для условия «содержит X»

Заведём два массива:

- `dp_no[i]` — количество путей из A в i, **не** проходящих через X.
- `dp_yes[i]` — количество путей из A в i, **проходящих** через X.

```
dp_no = [0] * (end + 1)
dp_yes = [0] * (end + 1)
dp_no[A] = 1

for i in range(A, end + 1):
    if i in forbidden_set:
        continue
    for step in moves:
        if i + step > end:
            continue
        nxt = i + step
        if nxt == X:
            dp_yes[nxt] += dp_no[i]  # Достигли X впервые
        else:
            dp_no[nxt] += dp_no[i]
        dp_yes[nxt] += dp_yes[i]
```

После заполнения ответ: `dp_yes[end]`.

6. Практика

Задача

Условие:

Исполнитель: +1, +2, +3, +4, +5. Сколько программ из 1 в 100, содержащих число 7 и не содержащих 56?

Анализ:

- Содержит 7 → разбиваем на два этапа: 1→7 и 7→100.
- Не содержит 56 → на обоих этапах запрещаем 56.

Решение 1: рекурсия с разбиением

```
from functools import lru_cache

moves = [1, 2, 3, 4, 5]
FORBIDDEN = 56
REQUIRED = 7
```

```python
@lru_cache(maxsize=None)
def paths(start, end):
    if start == end:
        return 1
    if start > end or start == FORBIDDEN:
        return 0
    return sum(paths(start + step, end) for step in moves)

result = paths(1, REQUIRED) * paths(REQUIRED, 100)
print(result)
```

Решение 2: восходящее ДП с двумя массивами

```python
A, B = 1, 100
REQUIRED = 7
FORBIDDEN = 56
moves = [1, 2, 3, 4, 5]

dp_no = [0] * (B + 1)   # не проходили через 7
dp_yes = [0] * (B + 1)  # прошли через 7
dp_no[A] = 1

for i in range(A, B + 1):
    if i == FORBIDDEN:
        dp_no[i] = dp_yes[i] = 0  # обнуляем запрещённую клетку
        continue
    for step in moves:
        nxt = i + step
        if nxt > B:
            continue
        if nxt == REQUIRED:
            dp_yes[nxt] += dp_no[i]
        else:
            dp_no[nxt] += dp_no[i]
        dp_yes[nxt] += dp_yes[i]

print(dp_yes[B])
```

Оба решения дают одинаковый ответ.

Сводка методов для сложного №23

Условие	Метод
Содержит X	Разбиение: `paths(A,X) * paths(X,B)`
Не содержит Y	Параметр `forbidden` в рекурсии или проверка в цикле
Содержит X и не содержит Y	Разбиение + `forbidden` на обоих этапах
Много условий, большой диапазон	Восходящее ДП с несколькими массивами

Домашнее задание

Задание 1

Исполнитель: +1, +2, ×2. Сколько программ из 2 в 40, содержащих 10 и не содержащих 30?

Задание 2

Исполнитель: +1, +3, +5. Сколько программ из 1 в 80, содержащих 20 и не содержащих 40 и 60?

Задание 3

Решите задание 2 восходящим ДП с двумя массивами. Сравните ответы.

Что мы сегодня изучили

Тема	Ключевые моменты
Содержит X	Разбиение на два этапа, умножение
Не содержит Y	Флаг `forbidden` в рекурсии или проверка в ДП
Комбинация	Разбиение + запрет на обоих этапах
Восходящее ДП	Два массива (`dp_no` , `dp_yes`), нет рекурсии
Выбор метода	Рекурсия — для малых диапазонов, восходящее — для больших

> **Связь с ЕГЭ:** Задание №23 — 1 балл. Сложные условия решаются комбинацией разбиения и восходящего ДП.

На следующем занятии (Занятие 27)

Тема: ДП по остаткам (задание №27)

- • Максимальная сумма пары/тройки с условиями
- • Хранение лучших значений по остаткам
- • Один проход по массиву: O(N)

Чек-лист занятия

- ☐ Умею решать «содержит X» разбиением на два этапа
- ☐ Умею добавлять запрет числа в рекурсию и восходящее ДП
- ☐ Знаю, как комбинировать оба условия
- ☐ Могу реализовать восходящее ДП с двумя массивами
- ☐ Выполнено домашнее задание

Занятие 27.1. Сложные задачи №27 — все типы

План занятия

1. Классификация всех типов задач №27
2. Тип 1: Пары/тройки с кратностью (остатки)
3. Тип 2: Расстояние между элементами
4. Тип 3: Выбор из пар с корректировкой
5. Тип 4: Подпоследовательности с условием
6. Тип 5: Оптимальное размещение (линейная/кольцевая трасса)
7. Тип 6: Структурные (Si > Sj < Sk и подобные)
8. Разбор 59732, 68528, 69905, 38961, 33529, 47231, 33529, 61407, 58494, 33106, 59824, 59777

> **Цель занятия:** Охватить все разновидности задания №27 и научиться определять тип задачи по условию.

1. Классификация всех типов задач №27

Шесть основных типов

Тип	Описание	Ключевой метод
1. Кратность	Сумма/произведение пары или тройки кратно K	`best[остаток]`
2. Расстояние	Элементы на расстоянии ≥ K	Отложенное обновление (очередь)
3. Выбор из пар	Из каждой пары выбрать ровно одно число	Жадный выбор + корректировка разностями
4. Подпоследовательности	Непрерывный отрезок с условием на сумму/количество	Префиксные суммы + `best[остаток]`
5. Оптимальное размещение	Где разместить цех/лабораторию на трассе	Префиксные суммы + пересчёт за O(1)
6. Структурные	Si > Sj < Sk, возрастание/убывание	Префиксные/суффиксные максимумы/минимумы

Как определить тип по условию

- **«кратно K»**, **«остаток от деления»**, **«делится на»** → Тип 1.
- **«расстояние между элементами»**, **«не менее K»**, **«разность индексов»** → Тип 2.
- **«из каждой пары ровно одно число»**, **«выбрать из пары»** → Тип 3.
- **«непрерывная подпоследовательность»**, **«идущих подряд»**, **«участок»** → Тип 4.
- **«расположить лабораторию»**, **«пункты на трассе»**, **«цех подготовки»**, **«кольцевая»** → Тип 5.
- **«i < j < k»**, **«Si > Sj < Sk»**, **«три элемента с условием на порядок»** → Тип 6.

2. Тип 1: Пары/тройки с кратностью (остатки)

Идея метода

Если нужно, чтобы сумма пары делилась на K, то для элемента x с остатком r = x % K нужен партнёр с остатком (K − r) % K. Поэтому мы храним **лучший (максимальный или минимальный) элемент для каждого остатка**, встреченный ранее. При обработке очередного элемента x мы смотрим, есть ли в сохранённых лучших элементах партнёр с нужным остатком, и если есть — обновляем ответ.

Важно для произведения

Для **произведения** нужно хранить и **максимум**, и **минимум** для каждого остатка, потому что произведение двух отрицательных чисел даёт положительное, и максимум произведения может получиться из двух минимумов.

Сложность

O(N) для пары, O(N·K) для тройки.

3. Тип 2: Расстояние между элементами

Идея метода

Если элементы в паре (или тройке) должны отстоять друг от друга не менее чем на K позиций, то элемент, встреченный на позиции i, может быть использован как «первый» в паре только для элементов на позициях i+K и далее. Поэтому мы **откладываем** добавление элемента в «доступные» на K шагов.

Для **тройки** с расстоянием ≥ K между любыми двумя элементами:

- Первый элемент становится доступен как «первый в паре» сразу, но для второго нужно подождать K шагов.
- Пара становится доступна как «первые два в тройке» ещё через K шагов.

Сложность

O(N).

4. Тип 3: Выбор из пар с корректировкой

Идея метода

Даны пары чисел. Из каждой пары нужно выбрать **ровно одно** число. Сумма должна удовлетворять условию (чётность, кратность). Нужно максимизировать или минимизировать сумму.

Алгоритм:

1. Сначала делаем **жадный выбор**, который даёт экстремальную сумму без учёта условия (например, из каждой пары берём максимум для максимизации суммы).

2. Если условие не выполнено, нужно **заменить** некоторые числа на парные им. Замена меняет сумму на разность `новое - старое`.
3. Собираем все возможные «цены замены» (разности), которые меняют нужную характеристику (чётность, остаток).
4. Сортируем эти разности по возрастанию (для минимизации потери) и заменяем нужное количество.

Сложность

O(N log N) из-за сортировки разностей.

5. Тип 4: Подпоследовательности с условием

Идея метода

Нужна **непрерывная** подпоследовательность (отрезок) с максимальной (или минимальной) суммой, причём некоторое свойство отрезка (например, количество чётных чисел) должно быть кратно K.

Используем **префиксные суммы**: `pref[i]` — сумма первых i элементов. Сумма на отрезке `[L+1, R]` равна `pref[R] - pref[L]`.

Для свойства (например, количество чётных) тоже считаем префиксные количества: `cnt[i]` — количество чётных среди первых i элементов. Тогда количество чётных на отрезке = `cnt[R] - cnt[L]`.

Храним `best[r]` — **минимальную** префиксную сумму (для максимизации разности), при которой остаток количества чётных равен r. Тогда для каждого R с остатком r ответ = `pref[R] - best[r]`.

Сложность

O(N).

6. Тип 5: Оптимальное размещение (линейная/кольцевая трасса)

Идея метода

Есть N пунктов на прямой (или кольце). Для каждого известен «вес» — количество пробирок, комплектов, контейнеров. Нужно разместить центр (цех, лабораторию) в одном из пунктов так, чтобы суммарная стоимость доставки (вес × расстояние) была минимальна.

Алгоритм:

1. Сортируем пункты по координате (если они уже не упорядочены).
2. Вычисляем стоимость для размещения в первом пункте «в лоб».
3. При сдвиге центра из пункта i−1 в пункт i:
   - Пункты **левее** центра отодвигаются → стоимость растёт на `сумма_весов_слева × расстояние_сдвига`.
   - Пункты **правее** (включая сам новый центр) приближаются → стоимость падает на `сумма_весов_справа × расстояние_сдвига`.

4. Так пересчитываем стоимость за O(1) для каждого положения центра.

Для кольцевой трассы расстояние считается по кратчайшему пути: `min(|i-j|, N - |i-j|)`. Решение усложняется: нужно отдельно считать для левой и правой половин кольца.

Сложность

O(N) после сортировки (или O(N), если пункты уже упорядочены).

7. Тип 6: Структурные (Si > Sj < Sk и подобные)

Идея метода

Нужно найти три элемента с индексами i < j < k, удовлетворяющие условиям на значения (например, Si > Sj и Sk > Sj), и максимизировать некоторое выражение от них.

Метод: Префиксные и суффиксные максимумы/минимумы.

`max_left[i]` — максимум среди элементов **левее** i (индексы < i). `max_right[i]` — максимум среди элементов **правее** i (индексы > i).

Тогда для каждого j (средний элемент) мы знаем лучшего кандидата слева и справа. Проверяем условия и обновляем ответ.

Сложность

O(N).

8. Решения

Задача 1

Условие (кратко): В текстовом файле содержится некоторое количество натуральных чисел. Первая строка — число K (минимальное расстояние), вторая — N (количество чисел). Определите максимальную сумму трёх чисел, чтобы **любые два** числа находились на расстоянии не менее K друг от друга.

Тип: 2 (расстояние).

Решение:

```
K = int(input())
N = int(input())
data = [int(input()) for _ in range(N)]

best1 = -10**18  # лучший первый элемент (на K левее второго)
best2 = -10**18  # лучшая сумма первых двух (на K левее третьего)
max_sum = -10**18

for i in range(2 * K, N):
    best1 = max(best1, data[i - 2 * K])
    best2 = max(best2, best1 + data[i - K])
```

```
    max_sum = max(max_sum, best2 + data[i])

print(max_sum)
```

Задача 2

Условие (кратко): Кольцевая трасса длины N километров. На каждом километре — пункт питания с известным количеством комплектов. Стоимость доставки = количество комплектов × расстояние от мобильного цеха до пункта. Цех размещается в одном из пунктов. Найти минимальную суммарную стоимость доставки во все пункты.

Тип: 5 (кольцевая трасса).

Решение:

```
N = int(input())
amounts = [int(input()) for _ in range(N)]

# Стоимость для цеха в пункте 0 (расстояние по кольцу)
total_cost = 0
for i in range(N):
    dist = min(i, N - i)
    total_cost += amounts[i] * dist

min_cost = total_cost

# Перебираем положение цеха
# При сдвиге цеха вправо: левая половина отодвигается, правая приближается
left_sum = 0
right_sum = sum(amounts)

for center in range(1, N):
    # Элемент center-1 переходит из правой половины в левую
    left_sum += amounts[center - 1]
    right_sum -= amounts[center - 1]
    # Стоимость меняется: левые +1, правые -1
    total_cost = total_cost + left_sum - right_sum
    min_cost = min(min_cost, total_cost)

print(min_cost)
```

> Это упрощённая версия для **линейной** трассы. Для кольцевой нужно учитывать расстояние как `min(|i-j|, N−|i-j|)`, что требует более сложного пересчёта.

Задача 3

Условие (кратко): Дана последовательность S из N чисел. Найдите три элемента Si, Sj, Sk с индексами i < j < k, такие что Si > Sj и Sk > Sj. Значение выражения (Si − Sj) + (Sk − Sj) должно быть максимально.

Тип: 6 (структурная).

Решение:

```
N = int(input())
data = [int(input()) for _ in range(N)]
```

```
max_left = [-10**18] * N
cur = -10**18
for i in range(N):
    max_left[i] = cur
    cur = max(cur, data[i])

max_right = [-10**18] * N
cur = -10**18
for i in range(N - 1, -1, -1):
    max_right[i] = cur
    cur = max(cur, data[i])

best = -10**18
for j in range(1, N - 1):
    if max_left[j] > data[j] and max_right[j] > data[j]:
        best = max(best, max_left[j] + max_right[j] - 2 * data[j])

print(best)
```

Задача 4

Условие (кратко): Дана последовательность натуральных чисел. Найдите максимальную сумму **непрерывной подпоследовательности**, в которой **количество чётных элементов кратно k = 10**.

Тип: 4 (подпоследовательность).

Решение:

```
N = int(input())
K = 10
best = [10**18] * K  # минимальная префиксная сумма для остатка
best[0] = 0
max_sum = -10**18
pref = 0
even_count = 0

for _ in range(N):
    x = int(input())
    pref += x
    if x % 2 == 0:
        even_count += 1
    r = even_count % K
    if best[r] != 10**18:
        max_sum = max(max_sum, pref - best[r])
    best[r] = min(best[r], pref)

print(max_sum)
```

Задача 5

Условие (кратко): Набор данных состоит из **нечётного количества пар** натуральных чисел. Из каждой пары нужно выбрать ровно одно число. Чётность суммы выбранных чисел должна совпадать с чётностью **большинства** выбранных чисел. Найдите максимальную возможную сумму.

Тип: 3 (выбор из пар).

Решение:

```
N = int(input())
total = 0
odd_count = 0
diffs = []  # разности, замена которых меняет чётность

for _ in range(N):
    a, b = map(int, input().split())
    mx, mn = max(a, b), min(a, b)
    total += mx
    if mx % 2 == 1:
        odd_count += 1
    if (mx % 2) != (mn % 2):
        diffs.append(mx - mn)

diffs.sort()
majority_odd = odd_count > N - odd_count
sum_odd = total % 2 == 1

if sum_odd != majority_odd:
    total -= diffs[0]

print(total)
```

Задача 6

Условие (кратко): N пунктов приёма биоматериалов вдоль автомагистрали. Для каждого известен номер (расстояние от нуля) и количество пробирок. Пробирки перевозят в контейнерах вместимостью до 36 штук. Стоимость перевозки = расстояние от пункта до лаборатории × количество контейнеров. Лабораторию размещают в одном из пунктов. Найдите минимальную общую стоимость доставки.

Тип: 5 (линейная трасса с контейнерами).

Решение:

```
CAP = 36
N = int(input())
points = []
for _ in range(N):
    pos, amount = map(int, input().split())
    containers = (amount + CAP - 1) // CAP
    points.append((pos, containers))

points.sort()

total_cont = sum(c for _, c in points)
left_cont = 0
right_cont = total_cont

total_cost = 0
base = points[0][0]
for pos, c in points:
```

```
    total_cost += c * (pos - base)

min_cost = total_cost

for i in range(1, N):
    dist = points[i][0] - points[i-1][0]
    left_cont += points[i-1][1]
    right_cont -= points[i-1][1]
    total_cost = total_cost + left_cont * dist - right_cont * dist
    min_cost = min(min_cost, total_cost)

print(min_cost)
```

Задача 7

Условие (кратко): Аналогично задаче 5: нечётное количество пар, выбрать из каждой пары одно число. Чётность суммы должна совпадать с чётностью большинства. Максимизировать сумму.

Тип: 3 (выбор из пар).

Решение: Аналогично задаче 5.

Задача 8

Условие (кратко): Дана последовательность целых чисел. Расстояние между элементами — разность их индексов. Выберите три числа так, чтобы **максимальное** расстояние между выбранными числами было не меньше 2K, а их сумма была максимально возможной.

Тип: 2 (расстояние, вариант с максимальным расстоянием).

Решение:

```
K = int(input())
N = int(input())
data = [int(input()) for _ in range(N)]

# Нужно выбрать i < j < k так, что k - i ⩾ 2K
# best_first[i] — лучший элемент на расстоянии ⩾ 2K слева
best_first = [-10**18] * N
cur = -10**18
for i in range(N):
    if i ⩾ 2 * K:
        cur = max(cur, data[i - 2 * K])
    best_first[i] = cur

# best_middle[i] — лучшая сумма первого и среднего, где первый отстоит ⩾
2K, средний — между ними
best_middle = [-10**18] * N
cur = -10**18
for i in range(N):
    if i ⩾ K:
        cur = max(cur, best_first[i - K] + data[i - K])
    best_middle[i] = cur

max_sum = -10**18
```

```
for i in range(2 * K, N):
    if best_middle[i] != -10**18:
        max_sum = max(max_sum, best_middle[i] + data[i])

print(max_sum)
```

Задача 9

Условие (кратко): Дана последовательность натуральных чисел. Назовём парой любые два числа из последовательности. Определите **количество пар**, в которых **сумма элементов и расстояние между ними имеют равные остатки от деления на 9**.

Тип: 1 (кратность, подсчёт количества).

Решение:

```
N = int(input())
data = [int(input()) for _ in range(N)]

# Условие: (x + y) % 9 == (j - i) % 9
# ⇔ (x - i) % 9 == (-y - j) % 9
# ⇔ (x - i) % 9 == ((-y - j) % 9 + 9) % 9

count = [0] * 9
result = 0

for i, x in enumerate(data):
    r = ((x - i) % 9 + 9) % 9
    need = ((-x - i) % 9 + 9) % 9
    result += count[need]
    count[r] += 1

print(result)
```

Задача 10

Условие (кратко): Набор данных состоит из пар натуральных чисел. Из каждой пары нужно выбрать ровно одно число так, чтобы **сумма всех выбранных чисел делилась на 3** и была **минимально возможной**.

Тип: 3 (выбор из пар, минимизация с кратностью).

Решение:

```
N = int(input())
total = 0
diffs = [[] for _ in range(3)]  # разности по остаткам

for _ in range(N):
    a, b = map(int, input().split())
    mn, mx = min(a, b), max(a, b)
    total += mn
    diff = mx - mn
    r = diff % 3
    diffs[r].append(diff)
```

```
for r in range(3):
    diffs[r].sort()

r = total % 3
if r ≠ 0:
    need = (3 - r) % 3
    best = 10**18
    # Одна замена
    if diffs[need]:
        best = min(best, diffs[need][0])
    # Две замены
    for r1 in range(3):
        for r2 in range(3):
            if (r1 + r2) % 3 == need:
                if r1 == r2 and len(diffs[r1]) ≥ 2:
                    best = min(best, diffs[r1][0] + diffs[r1][1])
                elif r1 ≠ r2 and diffs[r1] and diffs[r2]:
                    best = min(best, diffs[r1][0] + diffs[r2][0])
    total += best

print(total)
```

Задача 11

Условие (кратко): Геодезист измеряет высоту для каждой из N метровых отметок дороги. Нужно выбрать **участок дороги длиной не менее K метров**, на котором **сумма высот максимальна**. Найти эту максимальную сумму.

Тип: 4 (подпоследовательность с минимальной длиной).

Решение:

```
N = int(input())
K = int(input())
data = [int(input()) for _ in range(N)]

pref = [0] * (N + 1)
for i in range(N):
    pref[i + 1] = pref[i] + data[i]

max_sum = -10**18
min_pref = 10**18

for i in range(K, N + 1):
    min_pref = min(min_pref, pref[i - K])
    max_sum = max(max_sum, pref[i] - min_pref)

print(max_sum)
```

Задача 12

Условие (кратко): Дана последовательность из N натуральных чисел и число K — минимальное расстояние. Найдите **минимальное значение произведения трёх элементов** так, что между любыми двумя элементами тройки расстояние не менее K.

Тип: 2 (расстояние + произведение).

Решение:

```
N = int(input())
K = int(input())
data = [int(input()) for _ in range(N)]

INF = 10**18
best1_min = INF
best1_max = -INF
best2_min = INF
best2_max = -INF
min_prod = INF

for i in range(2 * K, N):
    # Обновляем best1
    x = data[i - 2 * K]
    best1_min = min(best1_min, x)
    best1_max = max(best1_max, x)

    # Обновляем best2 (произведение первого и второго)
    y = data[i - K]
    if abs(best1_min) != INF:
        p1 = best1_min * y
        best2_min = min(best2_min, p1)
        best2_max = max(best2_max, p1)
    if abs(best1_max) != INF:
        p2 = best1_max * y
        best2_min = min(best2_min, p2)
        best2_max = max(best2_max, p2)

    # Проверяем тройку
    z = data[i]
    if abs(best2_min) != INF:
        min_prod = min(min_prod, best2_min * z)
    if abs(best2_max) != INF:
        min_prod = min(min_prod, best2_max * z)

print(min_prod)
```

Сводная таблица всех 12 задач

№	Тип	Суть задачи
1	Расстояние	Максимальная сумма трёх чисел, расстояние между любыми ≥ K
2	Кольцевая трасса	Минимальная стоимость доставки по кольцу
3	Структурная	Si > Sj < Sk, максимизировать (Si−Sj)+(Sk−Sj)
4	Подпоследовательность	Максимальная сумма отрезка, чётных кратно 10
5	Выбор из пар	Чётность суммы = чётность большинства, максимум
6	Линейная трасса	Контейнеры, минимальная стоимость доставки
7	Выбор из пар	Аналог задачи 5
8	Расстояние	Три числа, максимальное расстояние ≥ 2K, макс. сумма
9	Кратность	Количество пар: (сумма) % 9 = (расстояние) % 9
10	Выбор из пар	Минимальная сумма, кратная 3
11	Подпоследовательность	Участок ≥ K метров, максимальная сумма высот
12	Расстояние + произведение	Минимальное произведение трёх, расстояние ≥ K

Домашнее задание

Задание 1

Реализуйте решение задачи 2 (кольцевая трасса) полностью, с учётом кратчайшего расстояния по кольцу.

Задание 2

Решите задачу 9 (количество пар с равными остатками) для обоих файлов A и B.

Задание 3

Решите задачу 12 (минимальное произведение тройки) для обоих файлов A и B.

Что мы сегодня изучили

Тема	Ключевые моменты
Классификация	6 типов задач №27
Тип 1: Кратность	`best[остаток]`, для произведения — минимум и максимум
Тип 2: Расстояние	Отложенное обновление, «созревание» элементов
Тип 3: Выбор из пар	Жадный выбор + корректировка отсортированными разностями
Тип 4: Подпоследовательности	Префиксные суммы + `best[остаток]`
Тип 5: Размещение	Пересчёт стоимости за O(1) при сдвиге центра
Тип 6: Структурные	Префиксные/суффиксные максимумы/минимумы

> **Связь с ЕГЭ:** Задание №27 — 2 балла. Все 6 типов решаются за O(N) с правильным выбором структуры данных.

Чек-лист занятия

- ☐ Знаю 6 типов задач №27
- ☐ Умею определять тип задачи по условию
- ☐ Могу реализовать каждый тип
- ☐ Понимаю, как пересчитывать стоимость за O(1)
- ☐ Знаю, как корректировать сумму разностями
- ☐ Выполнено домашнее задание

Занятие 27.2. Кластеризация (задание №27, второй тип)

План занятия

1. Что такое кластеризация в ЕГЭ
2. Как определить количество кластеров и разделить точки
3. Центр кластера: определение и поиск
4. Радиус и диаметр кластера
5. Аномалии (лишние точки): как исключать
6. Разбор 75264, 76242, 76421, 76437, 84721, 81811

> **Цель занятия:** Освоить задачи на кластеризацию — второй тип задания №27. Это 2 первичных балла.

1. Что такое кластеризация в ЕГЭ

Суть задачи

Дан набор точек на плоскости (координаты x, y). Точки **естественным образом** разбиты на несколько групп — **кластеров**. Кластеры визуально отделены друг от друга: расстояние между точками разных кластеров значительно больше, чем между точками одного кластера.

Что нужно сделать:

1. Разделить точки на кластеры.
2. Для каждого кластера найти **центр** — точку кластера с минимальной суммой расстояний до остальных точек кластера.
3. Вычислить характеристики: радиус, диаметр, координаты центров и т.д.

Характеристики кластера

Термин	Определение
Центр (центроид)	Точка кластера, сумма расстояний от которой до всех остальных точек кластера **минимальна**
Радиус	Максимальное расстояние от центра до точек кластера
Диаметр	Максимальное расстояние между любыми двумя точками кластера

Расстояние между точками

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

В большинстве задач **не обязательно извлекать корень** — можно работать с квадратами расстояний, что экономит время и избегает погрешностей. Но в некоторых задачах (радиус, диаметр) корень нужен для финального ответа.

2. Как определить количество кластеров и разделить точки

Визуальный подход (основной)

Построить график по данным из файла (в Excel или Python `matplotlib`). Глазами видно, сколько кластеров и где они находятся.

Автоматический подход

Если файл большой (10 000 точек), можно разделить кластеры **по координате X или Y**: найти «провалы» в гистограмме.

Алгоритм:

1. Отсортировать точки по X.
2. Найти промежутки, где расстояние между соседними точками **резко возрастает** (например, в 5–10 раз больше среднего).
3. Разрезать по этим промежуткам.

```
points.sort()  # сортировка по X, затем по Y
clusters = []
current_cluster = [points[0]]

for i in range(1, len(points)):
    if points[i][0] - points[i-1][0] > THRESHOLD:
        clusters.append(current_cluster)
        current_cluster = []
    current_cluster.append(points[i])
clusters.append(current_cluster)
```

Подбор порога: вычислить среднее расстояние между соседними X, порог = среднее × 5.

Особые случаи

- Кластеры могут быть разделены по Y, а не по X — тогда сортируем по Y.
- Кластеры могут быть расположены по диагонали — тогда сортируем по X+Y или используем более сложные методы.
- В большинстве задач ЕГЭ кластеры разделены либо по X, либо по Y.

3. Центр кластера: определение и поиск

Определение

Центр кластера — это **одна из точек кластера**, для которой сумма расстояний до всех остальных точек кластера **минимальна**.

Алгоритм поиска

Перебираем каждую точку кластера как кандидата в центр. Для каждой считаем сумму квадратов расстояний (или самих расстояний) до всех остальных точек. Выбираем точку с минимальной суммой.

```
def find_center(cluster):
    best_point = None
    best_sum = float('inf')

    for p in cluster:
        total = 0
        for q in cluster:
            if p ≠ q:
                dist_sq = (p[0] - q[0])**2 + (p[1] - q[1])**2
                total += dist_sq  # или math.sqrt(dist_sq)
        if total < best_sum:
            best_sum = total
            best_point = p

    return best_point
```

Оптимизация

Для суммы **квадратов** расстояний центр совпадает с точкой, ближайшей к **среднему арифметическому** координат. Но по условию ЕГЭ центр — обязательно одна из точек кластера, поэтому ищем перебором.

Для файла B (до 10 000 точек) перебор $O(N^2)$ может быть долгим. Но в каждом **кластере** обычно не более 3000–5000 точек, что приемлемо.

4. Радиус и диаметр кластера

Радиус

После нахождения центра `C` радиус = максимальное расстояние от `C` до любой точки кластера.

```python
import math

def cluster_radius(cluster, center):
    max_dist = 0
    for p in cluster:
        dist = math.sqrt((p[0] - center[0])**2 + (p[1] - center[1])**2)
        max_dist = max(max_dist, dist)
    return max_dist
```

Диаметр

Диаметр = максимальное расстояние между **любыми двумя точками** кластера.

```python
def cluster_diameter(cluster):
    max_dist = 0
    best_pair = None
    for i in range(len(cluster)):
        for j in range(i + 1, len(cluster)):
            dist = math.sqrt((cluster[i][0] - cluster[j][0])**2 +
                             (cluster[i][1] - cluster[j][1])**2)
            if dist > max_dist:
                max_dist = dist
                best_pair = (cluster[i], cluster[j])
    return max_dist, best_pair
```

Для 5000 точек это ~12.5 млн пар — терпимо (1–2 секунды).

5. Аномалии (лишние точки): как исключать

Что такое аномалии

В некоторых задачах (обычно файл Б) есть 3 «лишние» точки — результат помех. Они **не принадлежат ни одному кластеру** и находятся далеко от всех остальных точек.

Как найти и исключить

После разделения на кластеры, для каждого кластера считаем его **размеры** (размах по X и Y). Точки, которые находятся далеко от «основной массы» — аномалии.

Простой способ:

1. Разделить все точки на кластеры описанным выше методом (по разрывам).
2. Очень маленькие «кластеры» (1–3 точки) — вероятно, аномалии.
3. Или: после нахождения центров, точки на расстоянии > 3× радиус от центра — аномалии.

```python
# Удаляем кластеры размером ⩽ 3 (аномалии)
clusters = [c for c in clusters if len(c) > 3]
```

6. Разбор всех 6 задач с полными условиями

Задача 1

Условие:

Результат испытания — точка (x, y). Проводится кластеризация: выделяются круги радиуса ≤ 3 так, что каждая точка попадает ровно в один кластер. Центр кластера — точка кластера с минимальным **максимальным** расстоянием до остальных точек. Радиус кластера — это максимальное расстояние от центра до точек кластера.

Обработка:

1. Исключить кластер с наибольшим числом точек.
2. Для оставшихся найти центры и радиусы.
3. Вычислить **средний радиус**.

Ответ: целая часть от (средний радиус × 10 000).

Алгоритм:

1. Разделить точки на кластеры (по разрывам в X или Y).
2. Найти кластер с максимальным количеством точек, исключить его.
3. Для каждого оставшегося кластера:
   - Найти центр (перебором, минимизируя **максимальное** расстояние до других точек).
   - Вычислить радиус = макс. расстояние от центра до точек.
4. Средний радиус = среднее арифметическое радиусов.
5. Ответ = `int(средний_радиус * 10000)`.

```
import math

# Чтение данных
points = []
with open("27_A.txt") as f:
    for line in f:
        x, y = map(float, line.split())
        points.append((x, y))

# Разделение на кластеры по разрывам в X
points.sort()
clusters = []
current = [points[0]]
for i in range(1, len(points)):
    if points[i][0] - points[i-1][0] > 1.0:  # порог подбирается
        clusters.append(current)
        current = []
    current.append(points[i])
clusters.append(current)

# Исключаем кластер с макс. количеством точек
```

```
clusters.sort(key=len, reverse=True)
clusters = clusters[1:]  # убираем самый большой

radii = []
for cluster in clusters:
    # Ищем центр: точка с минимальным МАКСИМАЛЬНЫМ расстоянием
    best_center = None
    best_max_dist = float('inf')
    for p in cluster:
        max_dist = 0
        for q in cluster:
            if p ≠ q:
                dist = math.sqrt((p[0]-q[0])**2 + (p[1]-q[1])**2)
                max_dist = max(max_dist, dist)
        if max_dist < best_max_dist:
            best_max_dist = max_dist
            best_center = p
    radii.append(best_max_dist)

avg_radius = sum(radii) / len(radii)
print(int(avg_radius * 10000))
```

Задача 2

Условие:

Звёзды на карте. Кластер — набор звёзд. Центр кластера — звезда, сумма расстояний от которой до всех остальных звёзд кластера минимальна.

Файл 27А: два кластера (≤ 1000 точек). Файл 27Б: три кластера (≤ 10 000 точек).

Для каждого файла найти центры кластеров. Вычислить:

- Px — среднее арифметическое абсцисс центров.
- Py — среднее арифметическое ординат центров.

Ответ: целая часть от (Px × 10 000) и целая часть от (Py × 10 000).

```
import math

def process_file(filename):
    points = []
    with open(filename) as f:
        for line in f:
            x, y = map(float, line.split())
            points.append((x, y))

    # Разделение на кластеры
    points.sort()
    clusters = []
    current = [points[0]]
    for i in range(1, len(points)):
        if points[i][0] - points[i-1][0] > 1.0:
            clusters.append(current)
```

```
            current = []
        current.append(points[i])
    clusters.append(current)

    # Удаление аномалий (кластеры размером ⩽ 3)
    clusters = [c for c in clusters if len(c) > 3]

    centers = []
    for cluster in clusters:
        best_center = None
        best_sum = float('inf')
        for p in cluster:
            total = 0
            for q in cluster:
                if p ≠ q:
                    total += math.sqrt((p[0]-q[0])**2 + (p[1]-q[1])**2)
            if total < best_sum:
                best_sum = total
                best_center = p
        centers.append(best_center)

    Px = sum(c[0] for c in centers) / len(centers)
    Py = sum(c[1] for c in centers) / len(centers)
    return int(abs(Px) * 10000), int(abs(Py) * 10000)

print(*process_file("27_A.txt"))
print(*process_file("27_B.txt"))
```

Задача 3

Условие:

Точки разбиты на кластеры, каждый лежит внутри квадрата со стороной H. Квадраты не пересекаются.

Файл А: два кластера, H = 4.7, ≤ 1000 точек. Файл Б: три кластера, H = 4, ≤ 10 000 точек.

Центр — точка кластера с минимальной суммой расстояний до остальных. Найти Px и Py — средние координат центров.

Ответ: целая часть от (|Px| × 10 000) и целая часть от (|Py| × 10 000).

Решение: Аналогично задаче 2, но порог для разделения кластеров может быть другим (зависит от H).

Задача 4

Условие:

Аналогично задаче 3, но другие H: Файл А: H = 4.7. Файл Б: H = 5.

Решение: То же самое, что и задача 3. Меняются только файлы и значения H.

Задача 5

Условие:

Кластеры лежат внутри прямоугольников H × W (не обязательно параллельных осям).

Файл А: два кластера, H = 3, W = 4, ≤ 1000 точек. Файл Б: три кластера, H = 6, W = 5, ≤ 10 000 точек. Есть ровно 3 аномальные точки.

Для файла А:

- Найти пары точек, образующие **диаметр** каждого кластера (макс. расстояние).
- Px — **максимальная** из сумм абсцисс этих точек по всем кластерам.
- Py — **максимальная** из сумм ординат этих точек.

Для файла Б:

- Q1 — диаметр кластера с **максимальным количеством точек**.
- Q2 — максимальное расстояние от точки диаметра одного кластера до точки диаметра другого кластера.

Ответ: целые части от (Px × 10 000), (Py × 10 000), (Q1 × 10 000), (Q2 × 10 000).

```
import math

def cluster_diameter_pair(cluster):
    max_dist = 0
    pair = None
    for i in range(len(cluster)):
        for j in range(i+1, len(cluster)):
            d = math.sqrt((cluster[i][0]-cluster[j][0])**2 +
                          (cluster[i][1]-cluster[j][1])**2)
            if d > max_dist:
                max_dist = d
                pair = (cluster[i], cluster[j])
    return max_dist, pair

# Для файла А:
pairs = [cluster_diameter_pair(c)[1] for c in clusters]
Px = max(p[0][0] + p[1][0] for p in pairs)
Py = max(p[0][1] + p[1][1] for p in pairs)
print(int(abs(Px) * 10000), int(abs(Py) * 10000))

# Для файла Б:
clusters_sorted = sorted(clusters, key=len, reverse=True)
largest = clusters_sorted[0]
Q1, _ = cluster_diameter_pair(largest)

# Точки диаметров для всех кластеров
diameter_points = []
for c in clusters:
    _, pair = cluster_diameter_pair(c)
    diameter_points.extend(pair)
```

```
Q2 = 0
for i in range(len(diameter_points)):
    for j in range(i+1, len(diameter_points)):
        d = math.sqrt((diameter_points[i][0]-diameter_points[j][0])**2 +
                      (diameter_points[i][1]-diameter_points[j][1])**2)
        Q2 = max(Q2, d)

print(int(Q1 * 10000), int(Q2 * 10000))
```

Задача 6

Условие:

Кластеры в прямоугольниках H × W.

Файл А: H = 6, W = 4.5, два кластера. Файл Б: H = 6, W = 5, три кластера, 3 аномалии.

Для файла А:

- Найти центры кластеров.
- Px — **минимальная** из абсцисс центров.
- Py — **минимальная** из ординат центров.

Для файла Б:

- Найти центры кластеров.
- Q1 — расстояние между центрами кластеров с **минимальным и максимальным** количеством точек.
- Q2 — **максимальное** расстояние от центра до точки этого же кластера среди всех кластеров.

Решение:

```
# Для файла А:
centers = [find_center(c) for c in clusters]
Px = min(c[0] for c in centers)
Py = min(c[1] for c in centers)
print(int(abs(Px) * 10000), int(abs(Py) * 10000))

# Для файла Б:
clusters_sorted = sorted(clusters, key=len)
min_cluster = clusters_sorted[0]
max_cluster = clusters_sorted[-1]
center_min = find_center(min_cluster)
center_max = find_center(max_cluster)
Q1 = math.sqrt((center_min[0]-center_max[0])**2 +
               (center_min[1]-center_max[1])**2)

Q2 = 0
for c in clusters:
    center = find_center(c)
    for p in c:
        d = math.sqrt((p[0]-center[0])**2 + (p[1]-center[1])**2)
        Q2 = max(Q2, d)
```

```
print(int(Q1 * 10000), int(Q2 * 10000))
```

Сводная таблица методов кластеризации

Шаг	Метод
Разделение на кластеры	Сортировка по X/Y, поиск разрывов
Центр кластера	Перебор точек, минимизация суммы расстояний
Радиус	Макс. расстояние от центра до точек
Диаметр	Макс. расстояние между любыми двумя точками
Аномалии	Исключение кластеров размером ≤ 3

Домашнее задание

Задание 1

Реализуйте разделение на кластеры для файла, где кластеры разделены по Y (а не по X).

Задание 2

Напишите функцию автоматического подбора порога для разделения кластеров.

Задание 3

Решите задачу 1 для обоих файлов A и B.

Что мы сегодня изучили

Тема	Ключевые моменты
Кластеризация	Разделение точек на группы
Разделение	Сортировка + поиск разрывов
Центр	Точка с минимальной суммой расстояний
Радиус/диаметр	Максимальные расстояния в кластере
Аномалии	Маленькие кластеры (≤ 3 точки)

> **Связь с ЕГЭ:** Кластеризация — второй тип задания №27 (2 балла). Ключевые навыки: разделение точек на кластеры и поиск центра.

Чек-лист занятия

- ☐ Понимаю, что такое кластеризация
- ☐ Умею разделять точки на кластеры по разрывам
- ☐ Знаю, как найти центр кластера перебором

- ☐ Могу вычислить радиус и диаметр
- ☐ Умею исключать аномалии
- ☐ Выполнено домашнее задание

Глава 7. Файлы и данные

Занятие 28. Чтение и обработка файлов (задание №17)

План занятия

1. Чтение данных из файла: `open()`, `readlines()`, `with`
2. Преобразование данных и фильтрация
3. Подсчёт количества, суммы, минимума/максимума
4. Парные и тройные условия (соседние элементы)
5. Практика

> **Цель занятия:** Научиться читать данные из файла и обрабатывать последовательности — задание №17 (1 первичный балл).

1. Чтение данных из файла

Основные способы

```
# Способ 1: with open (рекомендуется)
with open("17.txt") as f:
    data = [int(line.strip()) for line in f]

# Способ 2: readlines (весь файл в память)
with open("17.txt") as f:
    lines = f.readlines()
    data = [int(line.strip()) for line in lines]

# Способ 3: построчное чтение (экономия памяти)
data = []
with open("17.txt") as f:
    for line in f:
        data.append(int(line.strip()))
```

Конструкция `with open() as f:`

- Автоматически закрывает файл после выхода из блока.
- Не нужно вызывать `f.close()`.
- Работает даже если внутри блока произошла ошибка.

`strip()` — удаление пробельных символов

Каждая строка файла заканчивается символом `\n`. `strip()` удаляет его и пробелы по краям. Если не сделать `strip()`, `int()` может выдать ошибку.

2. Преобразование данных и фильтрация

Преобразование

```
int(line)      # строка → целое число
float(line)    # строка → дробное число
```

Фильтрация с условиями

```
# Элементы, кратные 3
filtered = [x for x in data if x % 3 == 0]

# Элементы, оканчивающиеся на 13
filtered = [x for x in data if x % 100 == 13]

# Трёхзначные числа
filtered = [x for x in data if 100 ⩽ x ⩽ 999]
```

3. Подсчёт количества, суммы, минимума/максимума

Базовые операции

```
count = len(data)          # количество
total = sum(data)          # сумма
min_val = min(data)        # минимум
max_val = max(data)        # максимум

# С условием:
count = sum(1 for x in data if условие)
total = sum(x for x in data if условие)
max_val = max(x for x in data if условие)
```

Осторожно с пустой последовательностью

```
# max([]) → ValueError!
# Правильно:
filtered = [x for x in data if условие]
if filtered:
    max_val = max(filtered)
```

4. Парные и тройные условия (соседние элементы)

Пара — два идущих подряд элемента

```
for i in range(len(data) - 1):
    a, b = data[i], data[i + 1]
    if условие(a, b):
        # обработка пары
```

Тройка — три идущих подряд элемента

```
for i in range(len(data) - 2):
    a, b, c = data[i], data[i + 1], data[i + 2]
    if условие(a, b, c):
        # обработка тройки
```

Пара — любые два различных элемента (не обязательно соседние)

Для небольших файлов (≤ 10 000) можно перебрать все пары:

```
for i in range(len(data)):
    for j in range(i + 1, len(data)):
        a, b = data[i], data[j]
        if условие(a, b):
            # обработка
```

Для больших файлов (N > 10 000) такой перебор слишком долгий (O(N²)). Тогда используют словари или массивы по остаткам.

5. Практика

Задача 1

Условие:

Последовательность из 10 000 целых положительных чисел (≤ 10 000). Найти:

- Количество **пар** (любые два различных элемента, порядок не важен), сумма которых **кратна 9**.
- Максимальную из сумм таких пар.

Анализ:

- Пар: C(10000, 2) ≈ 50 млн — перебор возможен, но долгий (~1–2 секунды).
- **Оптимизация:** группируем элементы по остаткам от деления на 9.

```
with open("17.txt") as f:
    data = [int(line) for line in f]

# Группируем по остаткам: для каждого остатка храним отсортированный
список
groups = [[] for _ in range(9)]
for x in data:
    groups[x % 9].append(x)

for r in range(9):
    groups[r].sort(reverse=True)

count = 0
max_sum = 0

for r1 in range(9):
```

```
    for r2 in range(r1, 9):
        if (r1 + r2) % 9 != 0:
            continue
        if r1 == r2:
            # Пары внутри одной группы
            n = len(groups[r1])
            count += n * (n - 1) // 2
            if n >= 2:
                max_sum = max(max_sum, groups[r1][0] + groups[r1][1])
        else:
            # Пары между разными группами
            count += len(groups[r1]) * len(groups[r2])
            if groups[r1] and groups[r2]:
                max_sum = max(max_sum, groups[r1][0] + groups[r2][0])

print(count, max_sum)
```

Объяснение:

- Если остатки r1 + r2 кратны 9, то любой элемент из группы r1 образует подходящую пару с любым элементом из группы r2.
- Для максимальной суммы берём самые большие элементы в группах.

Задача 2

Условие:

Последовательность целых чисел от −10 000 до 10 000. Найти количество **пар идущих подряд** элементов, в которых:

- Только одно число оканчивается на 3.
- Сумма квадратов элементов пары ≥ квадрата **максимального элемента последовательности, оканчивающегося на 3**.

Вывести: количество пар и максимальную из сумм квадратов таких пар.

Анализ:

- Сначала находим `max_ending_3` — максимальный элемент, оканчивающийся на 3.
- Затем проходим по парам соседей и проверяем условия.

```
with open("17.txt") as f:
    data = [int(line) for line in f]

# Максимальный элемент, оканчивающийся на 3
ending_3 = [x for x in data if abs(x) % 10 == 3]
max_ending_3 = max(ending_3)
target = max_ending_3 ** 2

count = 0
max_sq_sum = 0

for i in range(len(data) - 1):
    a, b = data[i], data[i + 1]
```

```
    # Только одно оканчивается на 3
    ends_3_a = abs(a) % 10 == 3
    ends_3_b = abs(b) % 10 == 3
    if ends_3_a != ends_3_b:  # Ровно одно
        sq_sum = a**2 + b**2
        if sq_sum >= target:
            count += 1
            max_sq_sum = max(max_sq_sum, sq_sum)

print(count, max_sq_sum)
```

Важно: `abs(x) % 10 == 3` проверяет последнюю цифру для отрицательных чисел тоже. −3 % 10 = 7 в Python, поэтому используем `abs()`.

Задача 3

Условие:

Последовательность натуральных чисел (≤ 100 000). Найти количество **троек идущих подряд** элементов, в которых:

- Ровно два из трёх — трёхзначные числа (от 100 до 999).
- Сумма тройки ≤ **максимального элемента последовательности, оканчивающегося на 13**.

Вывести: количество троек и максимальную сумму таких троек.

```
with open("17.txt") as f:
    data = [int(line) for line in f]

# Максимальный элемент, оканчивающийся на 13
ending_13 = [x for x in data if x % 100 == 13]
max_ending_13 = max(ending_13)

count = 0
max_sum = 0

for i in range(len(data) - 2):
    a, b, c = data[i], data[i + 1], data[i + 2]
    # Считаем количество трёхзначных
    three_digit_count = sum(1 for x in (a, b, c) if 100 <= x <= 999)
    if three_digit_count == 2:
        total = a + b + c
        if total <= max_ending_13:
            count += 1
            max_sum = max(max_sum, total)

print(count, max_sum)
```

Сводка методов для №17

Тип задачи	Метод
Пары (любые два)	Группировка по остаткам, O(N)
Пары (соседние)	Один проход `for i in range(N-1)`, O(N)
Тройки (соседние)	Один проход `for i in range(N-2)`, O(N)
Максимум с условием	Предварительная фильтрация, O(N)

Домашнее задание

Задание 1

Дана последовательность целых чисел. Найдите количество пар элементов (любых двух), произведение которых кратно 10, и максимальное произведение такой пары.

Задание 2

Дана последовательность. Найдите количество пар соседних элементов, где оба числа чётные, а их сумма больше среднего арифметического всей последовательности.

Задание 3

Дана последовательность. Найдите количество троек соседних элементов, где все три числа различны и их сумма кратна 6.

Что мы сегодня изучили

Тема	Ключевые моменты
Чтение файла	`with open() as f:`, `readlines()`, `strip()`
Фильтрация	`[x for x in data if условие]`
Пары (любые)	Группировка по остаткам для O(N)
Пары/тройки (соседние)	Один проход, O(N)
Максимум/минимум	`max()`, `min()` с условием

> **Связь с ЕГЭ:** Задание №17 — 1 балл. Решается за 5 минут чтением файла и одним проходом.

На следующем занятии (Занятие 29)

Тема: Сортировка и жадные алгоритмы (задание №26)

Чек-лист занятия

- ☐

Умею читать данные из файла

- ☐ Знаю, как фильтровать данные с условием
- ☐ Умею обрабатывать пары и тройки соседних элементов
- ☐ Знаю метод группировки по остаткам
- ☐ Выполнено домашнее задание

Занятие 29. Сортировка и жадные алгоритмы (задание №26)

План занятия

1. Сортировка в Python: `sort()`, `sorted()`, `key=lambda`
2. Жадные алгоритмы: когда работают
3. Два указателя для упаковки
4. Обработка событий во времени (парковка)
5. Практика

> **Цель занятия:** Освоить сортировку как инструмент и жадные алгоритмы для задания №26 (2 первичных балла).

1. Сортировка в Python

Основные методы

```
# list.sort() — сортирует список на месте, возвращает None
data.sort()

# sorted() — возвращает новый отсортированный список
new_data = sorted(data)

# reverse=True — по убыванию
data.sort(reverse=True)

# key= — сортировка по функции от элемента
data.sort(key=len)  # по длине строк
data.sort(key=lambda x: x[1])  # по второму элементу кортежа

# Сортировка по нескольким полям: кортеж в key
data.sort(key=lambda x: (x[0], -x[1]))  # по первому возр., второму убыв.
```

Типичные применения на ЕГЭ

- **Сортировка грузов по массе** для задачи об упаковке.

- **Сортировка событий по времени** для задачи о парковке.
- **Сортировка заявок по номеру дома и подъезда** для задачи о ремонте.

2. Жадные алгоритмы: когда работают

Что такое жадный алгоритм

На каждом шаге принимаем **локально оптимальное** решение, надеясь, что оно приведёт к глобальному оптимуму.

Когда работает

- Задача о **рюкзаке** с делимыми предметами.
- Задача об **упаковке в контейнеры** с двумя указателями.
- Задача о **выборе заявок** (по времени окончания).

Когда НЕ работает

- Задача о рюкзаке **0/1** (неделимые предметы, максимизация стоимости).
- Задачи, где нужно перебирать комбинации.

На ЕГЭ

Жадный алгоритм + сортировка — основной метод для №26.

3. Два указателя для упаковки

Идея

Есть набор предметов и контейнеры. Нужно упаковать максимальное количество предметов или использовать минимальное количество контейнеров.

Алгоритм (для грузовика):

1. Отсортировать грузы по возрастанию.
2. Один указатель `left` на самый лёгкий груз, другой `right` на самый тяжёлый.
3. Если `left + right ≤ грузоподъёмность` — упаковываем оба, сдвигаем оба указателя.
4. Иначе — упаковываем только `right`, сдвигаем `right`.

```
left, right = 0, len(weights) - 1
count = 0
total_weight = 0

while left ⩽ right:
    if weights[left] + weights[right] ⩽ CAPACITY:
        count += 2
        total_weight += weights[left] + weights[right]
        left += 1
```

```
        right -= 1
    else:
        count += 1
        total_weight += weights[right]
        right -= 1
```

4. Обработка событий во времени (парковка)

Идея

Автомобили приезжают и уезжают. Нужно отслеживать занятые места и освобождать их, когда время стоянки истекает.

Метод: храним **кучу** (heap) времён освобождения мест. Когда приезжает новый автомобиль, удаляем из кучи все времена освобождения, которые ≤ времени прибытия (места освободились).

```
import heapq

parking = []  # куча: (время_освобождения, тип_места)
free_A = 70
free_B = 30

for arrival, duration, car_type in events:
    # Освобождаем места
    while parking and parking[0][0] <= arrival:
        _, place_type = heapq.heappop(parking)
        if place_type == 'A':
            free_A += 1
        else:
            free_B += 1

    # Пытаемся припарковать
    if car_type == 'A':
        if free_A > 0:
            free_A -= 1
            heapq.heappush(parking, (arrival + duration, 'A'))
        elif free_B > 0:
            free_B -= 1
            heapq.heappush(parking, (arrival + duration, 'B'))
        else:
            # уезжает
            pass
    else:  # 'B'
        if free_B > 0:
            free_B -= 1
            heapq.heappush(parking, (arrival + duration, 'B'))
        else:
            # уезжает
            pass
```

5. Практика

Задача 1 (Ремонт подъездов)

Условие:

Дано N заявок: (номер_заявки, номер_дома, номер_подъезда). Всего > 100 домов. Найти дом с **наибольшим количеством подряд идущих подъездов**, на которые есть заявки. Если таких домов несколько — выбрать тот, где **наименьший искомый подъезд** имеет **максимальный номер заявки**.

Анализ:

1. Сгруппировать заявки по дому.
2. Для каждого дома найти подъезды с заявками, отсортировать.
3. Найти максимальную длину непрерывной цепочки подъездов.
4. При равенстве — по доп. условию.

```
with open("26.txt") as f:
    N = int(f.readline())
    applications = []
    for line in f:
        app_id, house, entrance = map(int, line.split())
        applications.append((app_id, house, entrance))

# Группируем по домам: дом → {подъезд: макс_номер_заявки}
from collections import defaultdict
houses = defaultdict(dict)
for app_id, house, entrance in applications:
    if entrance not in houses[house]:
        houses[house][entrance] = app_id
    else:
        houses[house][entrance] = max(houses[house][entrance], app_id)

best_house = None
best_len = 0
best_start_entrance = None
best_max_app_id = 0

for house, entrances_dict in houses.items():
    entrances = sorted(entrances_dict.keys())
    if not entrances:
        continue

    # Ищем максимальную непрерывную цепочку
    cur_start = entrances[0]
    cur_len = 1
    max_len = 1
    max_start = entrances[0]

    for i in range(1, len(entrances)):
        if entrances[i] == entrances[i-1] + 1:
            cur_len += 1
```

```
            if cur_len > max_len:
                max_len = cur_len
                max_start = cur_start
        else:
            cur_start = entrances[i]
            cur_len = 1

    if max_len > best_len:
        best_len = max_len
        best_house = house
        best_start_entrance = max_start
        best_max_app_id = entrances_dict[max_start]
    elif max_len == best_len:
        # Сравниваем по номеру заявки на наименьшем подъезде цепочки
        if entrances_dict[max_start] > best_max_app_id:
            best_house = house
            best_start_entrance = max_start
            best_max_app_id = entrances_dict[max_start]

print(best_house, best_start_entrance)
```

Задача 2 (Перевозка грузов)

Условие:

N грузов, грузоподъёмность M. Грузы 210–220 кг — обязательные, грузятся первыми. Остальное место заполняется максимальным количеством оставшихся грузов. При равном количестве — максимизируется масса самого тяжёлого, затем второго по величине и т.д.

Вывести: количество вывезенных грузов и их общую массу.

```
with open("c:/26.txt") as f:
    N, M = map(int, f.readline().split())
    weights = [int(line) for line in f]

# 1. Обязательные грузы
mandatory = [w for w in weights if 210 <= w <= 220]
optional = [w for w in weights if w < 210 or w > 220]

mandatory_count = len(mandatory)
mandatory_weight = sum(mandatory)
remaining = M - mandatory_weight

# 2. Сортируем
optional.sort()
n = len(optional)

# 3. Берём максимум самых лёгких
chosen = []
total = 0
for i in range(n):
    if total + optional[i] <= remaining:
        chosen.append(optional[i])
        total += optional[i]
    else:
```

```
        break

# 4. Улучшаем: заменяем выбранные на более тяжёлые, сохраняя количество
# Сортируем выбранные по возрастанию
chosen.sort()
unchosen = [w for w in optional if w not in chosen]

# Идём от самого тяжёлого из выбранных к самому лёгкому
for i in range(len(chosen) - 1, -1, -1):
    # Ищем самый тяжёлый из невыбранных, который можно поставить вместо
chosen[i]
    best_replace = chosen[i]
    for w in unchosen:
        if w > best_replace and total - chosen[i] + w ⩽ remaining:
            best_replace = w
    if best_replace ≠ chosen[i]:
        total = total - chosen[i] + best_replace
        unchosen.append(chosen[i])
        unchosen.remove(best_replace)
        chosen[i] = best_replace

total_count = mandatory_count + len(chosen)
total_weight = mandatory_weight + total
print(total_count, total_weight)
```

Задача 3 (Парковка)

Условие:

70 мест для легковых (A), 30 мест для микроавтобусов (B). Если мест A нет, легковой может занять место B. Микроавтобус не может занять место A.

Даны N событий: время прибытия, длительность стоянки, тип (A/B).

Вывести: количество припаркованных микроавтобусов, количество уехавших автомобилей.

```
import heapq

with open("26.txt") as f:
    N = int(f.readline())
    events = []
    for line in f:
        parts = line.split()
        arrival = int(parts[0])
        duration = int(parts[1])
        car_type = parts[2]
        events.append((arrival, duration, car_type))

events.sort()  # Сортируем по времени прибытия

free_A = 70
free_B = 30
parked_B = 0
left_count = 0

# Куча: (время_освобождения, тип_места)
```

```
parking_heap = []

for arrival, duration, car_type in events:
    # Освобождаем места, у которых время истекло
    while parking_heap and parking_heap[0][0] <= arrival:
        _, place_type = heapq.heappop(parking_heap)
        if place_type == 'A':
            free_A += 1
        else:
            free_B += 1

    if car_type == 'A':
        if free_A > 0:
            free_A -= 1
            heapq.heappush(parking_heap, (arrival + duration, 'A'))
        elif free_B > 0:
            free_B -= 1
            heapq.heappush(parking_heap, (arrival + duration, 'B'))
        else:
            left_count += 1
    else:  # 'B'
        if free_B > 0:
            free_B -= 1
            parked_B += 1
            heapq.heappush(parking_heap, (arrival + duration, 'B'))
        else:
            left_count += 1

print(parked_B, left_count)
```

Объяснение:

- Куча хранит время освобождения занятых мест.
- При приезде автомобиля удаляем из кучи все записи с временем ≤ текущего.
- Пытаемся припарковать по правилам.
- Считаем припаркованные B и уехавшие.

Сводка методов для №26

Тип задачи	Метод
Непрерывные подъезды	Группировка, сортировка, поиск цепочек
Упаковка в грузовик	Жадный алгоритм с последующим улучшением
Парковка	Обработка событий, куча (heap)

Домашнее задание

Задание 1

Решите задачу 1 (ремонт подъездов) для файла из банка ФИПИ.

Задание 2

Решите задачу 2 (грузовик) с полным учётом условия о максимизации массы самого тяжёлого груза при равном количестве.

Задание 3

Модифицируйте задачу 3 (парковка): добавьте третий тип мест C (для грузовиков, 10 мест), которые могут занимать только грузовики.

Что мы сегодня изучили

Тема	Ключевые моменты
Сортировка	`sort(key=lambda)`, по нескольким полям
Жадный алгоритм	Локально оптимальный выбор
Два указателя	Упаковка в контейнеры
Обработка событий	Куча (heap) для освобождения мест
Группировка	`defaultdict` для сбора данных по ключу

> **Связь с ЕГЭ:** Задание №26 — 2 балла. Ключевые навыки: сортировка, жадные алгоритмы, два указателя, обработка событий.

Далее

Excel и ручные задачи — задания №1, 3, 4, 7, 9, 10, 11, 13, 15, 18.

Чек-лист занятия

- ☐ Умею сортировать по нескольким полям
- ☐ Понимаю, когда жадный алгоритм работает
- ☐ Знаю метод двух указателей
- ☐ Умею обрабатывать события с кучей
- ☐ Выполнено домашнее задание

Глава 8. Excel/Calc и ручные задачи

Занятие 30. Задания №3 и №9 в LibreOffice (OpenOffice)

План занятия

1. Задание №3: три таблицы — объединение через ВПР и ИНДЕКС+ПОИСКПОЗ
2. Группировка данных: СУММЕСЛИМН, СЧЁТЕСЛИМН, СРЗНАЧЕСЛИМН
3. Сводные таблицы (Pivot Table) как альтернатива формулам
4. Задание №9: основные формулы для электронных таблиц
5. Практические примеры

> **Цель занятия:** Освоить LibreOffice Calc для решения заданий №3 и №9. Всё, что работает в LibreOffice, работает и в OpenOffice.

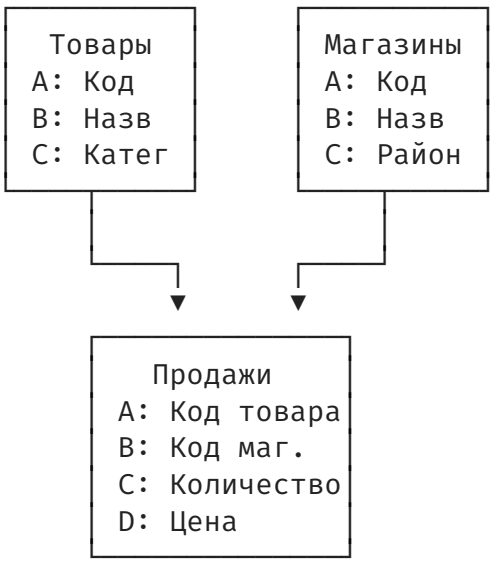
1. Задание №3: три таблицы — объединение через ВПР

Типичная структура данных

В задании №3 (усложнённый вариант) файл содержит **три листа**:

Лист	Содержит	Ключевой столбец
Товары	Код товара, Наименование, Категория	Код товара (A)
Магазины	Код магазина, Название, Район	Код магазина (A)
Продажи	Код товара, Код магазина, Количество, Цена	Код товара (A), Код магазина (B)

Связи между таблицами



Шаг 1. Подтягиваем категорию товара

На листе **Продажи** создаём столбец E «Категория». В ячейку E2 вводим:

```
=ВПР(A2; $Товары.$A$2:$C$10000; 3; 0)
```

Разбор формулы:

Аргумент	Значение	Пояснение
`A2`	Код товара из Продаж	Что ищем (ключ)
`$Товары.$A$2:$C$10000`	Таблица на листе «Товары»	Где ищем (первый столбец — ключи)
`3`	Номер столбца	Категория находится в 3-м столбце (A=1, B=2, C=3)
`0`	Тип поиска	0 = точное совпадение (обязательно для ЕГЭ!)

Почему знаки `$`: Они фиксируют диапазон, чтобы при копировании формулы вниз ссылка на таблицу товаров не смещалась.

После ввода формулы в E2 **протягиваем** её вниз на все строки продаж (хватаем за правый нижний угол ячейки и тянем).

Шаг 2. Подтягиваем район магазина

На листе **Продажи** создаём столбец F «Район». В ячейку F2 вводим:

```
=ВПР(B2; $Магазины.$A$2:$C$10000; 3; 0)
```

Разбор:

- `B2` — код магазина из Продаж.
- `$Магазины.$A$2:$C$10000` — таблица на листе «Магазины».
- `3` — район находится в 3-м столбце (A=код, B=название, C=район).

Протягиваем формулу вниз.

Шаг 3. Вычисляем сумму продажи

Создаём столбец G «Сумма». В G2:

```
=C2*D2
```

Протягиваем вниз. Теперь у нас есть всё для ответа.

2. Группировка данных: СУММЕСЛИМН, СЧЁТЕСЛИМН, СРЗНАЧЕСЛИМН

Теперь, когда все данные собраны на одном листе, можно отвечать на вопросы.

СУММЕСЛИМН — сумма по нескольким условиям

Вопрос: «Какова сумма продаж товаров категории **Канцелярия** в магазинах **Центрального** района?»

```
=СУММЕСЛИМН(G2:G10000; E2:E10000; "Канцелярия"; F2:F10000; "Центральный")
```

Синтаксис:

```
=СУММЕСЛИМН(диапазон_суммирования; диапазон_условия1; условие1;
диапазон_условия2; условие2; ...)
```

Аргумент	Значение
`G2:G10000`	Что суммируем (столбец Сумма)
`E2:E10000`	Где проверяем первое условие (Категория)
`"Канцелярия"`	Первое условие
`F2:F10000`	Где проверяем второе условие (Район)
`"Центральный"`	Второе условие

СЧЁТЕСЛИМН — количество по нескольким условиям

Вопрос: «Сколько продаж товаров категории **Напитки** в магазинах **Северного** района?»

```
=СЧЁТЕСЛИМН(E2:E10000; "Напитки"; F2:F10000; "Северный")
```

Синтаксис:

```
=СЧЁТЕСЛИМН(диапазон_условия1; условие1; диапазон_условия2; условие2; ...)
```

СРЗНАЧЕСЛИМН — среднее по нескольким условиям

Вопрос: «Какая средняя цена продажи товаров категории **Канцелярия**?»

```
=СРЗНАЧЕСЛИМН(D2:D10000; E2:E10000; "Канцелярия")
```

Синтаксис:

```
=СРЗНАЧЕСЛИМН(диапазон_усреднения; диапазон_условия1; условие1; ...)
```

> **Важно:** В LibreOffice разделитель аргументов — `;` (точка с запятой). В русской версии функции называются именно так: СУММЕСЛИМН, СЧЁТЕСЛИМН, СРЗНАЧЕСЛИМН.

3. Альтернатива ВПР: ИНДЕКС + ПОИСКПОЗ

Зачем нужна альтернатива

ВПР ищет **только в первом столбце** диапазона. Если ключ находится не в первом столбце, ВПР не сработает. ИНДЕКС+ПОИСКПОЗ лишены этого ограничения.

Синтаксис

```
=ИНДЕКС(диапазон_значений; ПОИСКПОЗ(ключ; диапазон_ключей; 0))
```

Пример

Подтянуть категорию товара (аналог ВПР):

```
=ИНДЕКС($Товары.$C$2:$C$10000; ПОИСКПОЗ(A2; $Товары.$A$2:$A$10000; 0))
```

Разбор:

Часть формулы	Что делает
`ПОИСКПОЗ(A2; $Товары.$A$2:$A$10000; 0)`	Находит **номер строки** в столбце A листа Товары, где значение равно A2
`ИНДЕКС($Товары.$C$2:$C$10000; ...)`	Возвращает значение из столбца C с тем же номером строки

ВПР vs ИНДЕКС+ПОИСКПОЗ

Критерий	ВПР	ИНДЕКС+ПОИСКПОЗ
Простота	Проще	Сложнее
Поиск не в первом столбце	Не может	Может
Скорость на больших таблицах	Медленнее	Быстрее
Для ЕГЭ	Достаточно	Избыточно, но работает

4. Сводные таблицы (Pivot Table) — альтернатива формулам

Что это

Сводная таблица позволяет группировать и суммировать данные **без формул** — перетаскиванием полей мышкой.

Как создать в LibreOffice

1. Выделите **всю таблицу** Продаж (с подтянутыми категорией и районом).
2. Меню: **Данные → Сводная таблица → Создать**.
3. В открывшемся диалоге:
   - Перетащите поле **«Категория»** в область **«Строки»**.
   - Перетащите поле **«Сумма»** в область **«Данные»**.
4. Нажмите **OK**.

Сводная таблица автоматически сгруппирует суммы по категориям.

Когда использовать

Ситуация	Что использовать
Нужен быстрый ответ	Сводная таблица
Нужно несколько разных агрегаций	Сводная таблица
Нужна воспроизводимость (проверка)	Формулы
Мало условий (1–2)	Формулы быстрее

5. Задание №9: электронные таблицы

Типичная структура

Одна таблица с данными. Например:

A (Дата)	B (Город)	C (Товар)	D (Количество)	E (Цена)
01.03	Москва	Ручка	100	30
02.03	Казань	Тетрадь	200	45
...	...	...	...	...

Основные формулы

Задача	Формула	Пример
Количество строк с условием	`=СЧЁТЕСЛИ(диапазон; условие)`	`=СЧЁТЕСЛИ(B2:B10000; "Москва")`
Количество строк, условие на число	`=СЧЁТЕСЛИ(диапазон; ">100")`	`=СЧЁТЕСЛИ(D2:D10000; "⩾50")`
Сумма по одному условию	`=СУММЕСЛИ(диап_усл; усл; диап_сумм)`	`=СУММЕСЛИ(B2:B10000; "Москва"; D2:D10000)`
Среднее по условию	`=СРЗНАЧЕСЛИ(диап_усл; усл; диап_ср)`	`=СРЗНАЧЕСЛИ(C2:C10000; "Ручка"; E2:E10000)`
Несколько условий (И)	`=СЧЁТЕСЛИМН( ... )` / `=СУММЕСЛИМН( ... )`	`=СУММЕСЛИМН(D2:D10000; B2:B10000; "Москва"; C2:C10000; "Ручка")`
Условие «ИЛИ»	Сложить отдельные СЧЁТЕСЛИ	`=СЧЁТЕСЛИ(B2:B10000; "Москва")+СЧЁТЕСЛИ(B2:B10000; "Казань")`
Максимум по условию	Через фильтр или доп. столбец	Отфильтровать → `=МАКС(диапазон)`
Минимум по условию	Аналогично	Отфильтровать → `=МИН(диапазон)`

Важные нюансы

- Условия-числа в кавычках: `">100"`, `"⩽50"`, `"<>0"` (не равно нулю).
- Условия-текст в кавычках: `"Москва"`.
- Если условие — ссылка на ячейку: `=СЧЁТЕСЛИ(B2:B10000; ">"&D1)`.

• СРЗНАЧЕСЛИ с одним диапазоном: третий аргумент можно опустить.

6. Полный пример решения №3 (три таблицы)

Условие

Лист Товары:

A (Код)	B (Наименование)	C (Категория)
1	Ручка	Канцелярия
2	Тетрадь	Канцелярия
3	Сок	Напитки

Лист Магазины:

A (Код)	B (Название)	C (Район)
10	Алые паруса	Центральный
20	Солнышко	Северный

Лист Продажи:

A (Код товара)	B (Код магазина)	C (Количество)	D (Цена)
1	10	5	30
2	20	10	45
3	10	8	60

Вопрос

«Какова сумма продаж товаров категории **Канцелярия** в магазинах **Центрального** района?»

Решение

Шаг 1. На листе Продажи, ячейка E2 (Категория):

```
=ВПР(A2; $Товары.$A$2:$C$10000; 3; 0)
```

Протянуть вниз.

Шаг 2. На листе Продажи, ячейка F2 (Район):

```
=ВПР(B2; $Магазины.$A$2:$C$10000; 3; 0)
```

Протянуть вниз.

Шаг 3. Ячейка G2 (Сумма):

```
=C2*D2
```

Протянуть вниз.

Шаг 4. В любой пустой ячейке:

```
=СУММЕСЛИМН(G2:G10000; E2:E10000; "Канцелярия"; F2:F10000; "Центральный")
```

Результат: 5 × 30 = 150 (продажа 1).

7. Полный пример решения №9

Условие

A (Дата)	B (Город)	C (Товар)	D (Количество)	E (Цена)
01.03.2024	Москва	Ручка	100	30
01.03.2024	Казань	Тетрадь	200	45
02.03.2024	Москва	Ручка	150	30
02.03.2024	Москва	Карандаш	80	25
03.03.2024	Казань	Ручка	120	30

Вопросы и ответы

1. Сколько продаж в Москве?

```
=СЧЁТЕСЛИ(B2:B10000; "Москва")
```

Ответ: 3.

2. Суммарное количество проданных ручек:

```
=СУММЕСЛИ(C2:C10000; "Ручка"; D2:D10000)
```

Ответ: 100 + 150 + 120 = 370.

3. Средняя цена товаров, проданных в количестве более 100:

```
=СРЗНАЧЕСЛИ(D2:D10000; ">100"; E2:E10000)
```

Строки с количеством > 100: (200, 45) и (150, 30) и (120, 30). Среднее: (45+30+30)/3 = 35.

4. Количество продаж ручек в Москве:

```
=СЧЁТЕСЛИМН(B2:B10000; "Москва"; C2:C10000; "Ручка")
```

Ответ: 2.

Шпаргалка по всем формулам

Формула	Назначение
`=ВПР(ключ; таблица; столбец; 0)`	Подтянуть значение из другой таблицы по ключу
`=ИНДЕКС(столбец; ПОИСКПОЗ(ключ; ключи; 0))`	То же, но ключ может быть в любом столбце
`=СУММЕСЛИ(условия; условие; суммы)`	Сумма по одному условию
`=СУММЕСЛИМН(суммы; усл1; зн1; усл2; зн2)`	Сумма по нескольким условиям
`=СЧЁТЕСЛИ(диапазон; условие)`	Количество по одному условию
`=СЧЁТЕСЛИМН(усл1; зн1; усл2; зн2)`	Количество по нескольким условиям
`=СРЗНАЧЕСЛИ(условия; условие; значения)`	Среднее по одному условию
`=СРЗНАЧЕСЛИМН(знач; усл1; зн1; ...)`	Среднее по нескольким условиям

Домашнее задание

Задание 1 (№3)

В файле три листа. Найдите сумму продаж товаров категории **«Продукты»** в магазинах **«Южного»** района.

Задание 2 (№9)

В таблице данные о продажах. Найдите:

- Количество продаж с ценой выше 100.
- Суммарную выручку по товару «Стол».
- Среднее количество товаров в продажах города «Казань».

Задание 3

Создайте сводную таблицу для данных из задания 2 и сравните результаты с формулами.

Что мы сегодня изучили

Тема	Ключевые моменты
ВПР	Объединение таблиц по ключу, только первый столбец
ИНДЕКС+ПОИСКПОЗ	Гибкое объединение, любой столбец
СУММЕСЛИМН и др.	Группировка с несколькими условиями
Сводные таблицы	Группировка без формул
№9: основные формулы	СЧЁТЕСЛИ, СУММЕСЛИ, СРЗНАЧЕСЛИ

> **Связь с ЕГЭ:** Задания №3 и №9 — 2 балла. Решаются в LibreOffice за 5–10 минут.

Чек-лист занятия

- ☐ Умею подтягивать данные из другой таблицы через ВПР
- ☐ Знаю альтернативу ИНДЕКС+ПОИСКПОЗ
- ☐ Умею использовать СУММЕСЛИМН, СЧЁТЕСЛИМН, СРЗНАЧЕСЛИМН
- ☐ Могу создать сводную таблицу
- ☐ Знаю формулы для задания №9
- ☐ Выполнено домашнее задание

Занятие 31. Ручные задачи: графы, сети, логика, ДП

План занятия

1. **Задание №1** — Граф и таблица: двухшаговый визуальный метод
2. **Задание №13** — Сети и маски на Python
3. **Задание №15** — Логика с отрезками (аналитический разбор)
4. **Задание №18** — Динамическое программирование в LibreOffice Calc

> **Цель занятия:** Освоить ручные и полуавтоматические методы для заданий №1, 13, 15, 18. Это 4 первичных балла.

1. Задание №1. Граф и таблица: двухшаговый визуальный метод

Суть задания

Дана **таблица расстояний** между пунктами (строки и столбцы пронумерованы) и **схема дорог** с пунктами, обозначенными **буквами** (А, Б, В, Г, Д, Е). Нужно определить, какой букве соответствует каждый номер в таблице.

Почему визуальный метод работает лучше

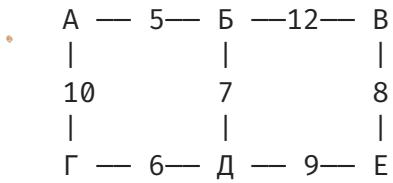
Человеческий мозг отлично распознаёт формы и трансформирует их в пространстве. Мы используем этот встроенный «процессор»: **рисуем** граф по таблице и **перерисовываем** его, пока он визуально не совпадёт с графом из условия.

Шаг 0. Что дано

Таблица (пример):

	П1	П2	П3	П4	П5	П6
П1	—	5	—	10	—	—
П2	5	—	7	—	12	—
П3	—	7	—	—	—	8
П4	10	—	—	—	6	—
П5	—	12	—	6	—	9
П6	—	—	8	—	9	—

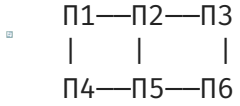
Схема из условия (буквы):



Шаг 1. Рисуем граф по таблице

1. Берём лист бумаги. Ставим 6 жирных точек — это вершины. Подписываем их **цифрами**: П1, П2, ..., П6.
2. Для каждого числа в таблице проводим линию (ребро) между соответствующими вершинами и подписываем на ней расстояние.
3. **Не заботимся о красоте** — просто соединяем точки.

Получится что-то вроде:



(это примерный вид, у вас будет свой)

Шаг 2. Первая перерисовка: убираем пересечения рёбер

1. Смотрим на свой рисунок. Если рёбра пересекаются (образуют «крест»), **перерисовываем** граф заново.
2. **Перетаскиваем** вершины в новые положения так, чтобы линии не перекрещивались.
3. Рёбра «тянутся» за вершинами — расстояния на них остаются те же.
4. Цель: получить **планарную укладку** — граф без пересечений.

> **Совет:** Начните с вершины, у которой больше всего соседей — поместите её в центр. Остальные расположите вокруг.

Шаг 3. Вторая перерисовка: приводим к виду из условия

1. Кладём рядом свой граф (уже без пересечений) и схему из условия (с буквами).
2. Сравниваем **форму**: расположение вершин, длины рёбер, количество соседей у каждой вершины.

3. **Перерисовываем** свой граф так, чтобы его вершины расположились **в том же порядке**, что и буквы на схеме.
4. Когда форма совпала — вершины вашего графа «сели» на места букв.

Пример:

- Если на схеме буква А находится слева вверху и соединена с двумя вершинами — ищите в своём графе вершину с двумя соседями, от которой отходят рёбра нужной длины.
- Перетащите эту вершину на место А. Остальные «подтянутся».

Шаг 4. Записываем ответ

Когда визуальное совпадение достигнуто, смотрим: П1 оказалась на месте буквы А, П2 — на месте Б, и т.д.

Ответ: последовательность букв в порядке номеров П1, П2, П3, П4, П5, П6.

Например: `АБВГДЕ`.

> **Главное правило:** Не пытайтесь решить в уме. Нарисуйте. Перерисуйте. Визуальное совпадение — самый надёжный способ.

2. Задание №13. Сети и маски (Python)

Суть задания

Даны IP-адреса и маска подсети. Нужно:

- Определить, находятся ли два узла в одной сети.
- Найти количество адресов в сети.
- Найти номер компьютера в сети.

Теория (минимально необходимая)

- **IP-адрес** — 32 бита (4 октета по 8 бит).
- **Маска подсети** — тоже 32 бита: сначала `k` единиц, потом `32-k` нулей.
- **Номер сети** = IP-адрес **побитовое И** с маской.
- **Номер компьютера** = IP-адрес **побитовое И** с инвертированной маской (~маска).
- **Количество адресов в сети** = `2^{32 - k}`, где `k` — количество единиц в маске.

Инструменты Python

```
# Перевод IP-строки в целое число
def ip_to_int(ip_str):
    parts = [int(x) for x in ip_str.split('.')]
    return (parts[0] << 24) | (parts[1] << 16) | (parts[2] << 8) |
parts[3]
```

```python
# Перевод целого числа в IP-строку
def int_to_ip(n):
    return f"{(n >> 24) & 255}.{(n >> 16) & 255}.{(n >> 8) & 255}.{n & 255}"

# Подсчёт единиц в двоичной записи числа
def count_ones(n):
    return bin(n).count('1')
```

Пример

Условие: Два узла: `192.168.1.10` и `192.168.1.55`. Маска: `255.255.255.0` (/24).

```python
ip1 = ip_to_int("192.168.1.10")
ip2 = ip_to_int("192.168.1.55")
mask = ip_to_int("255.255.255.0")

network1 = ip1 & mask
network2 = ip2 & mask

print("В одной сети" if network1 == network2 else "В разных сетях")
# В одной сети

# Номер компьютера
computer_num = ip2 & (~mask)
print(f"Номер компьютера: {computer_num}")  # 55

# Количество адресов
ones = count_ones(mask)
zeros = 32 - ones
print(f"Адресов в сети: {2 ** zeros}")  # 256
```

Варианты задач

Что дано	Что найти	Метод
IP, маска	Номер сети	`ip & mask`
IP, маска	Номер компьютера	`ip & ~mask`
Маска (или /k)	Количество адресов	`2 ** (32 - k)`
Два IP, маска	В одной сети?	`(ip1 & mask) == (ip2 & mask)`
IP, номер сети	Маска?	Подбор или вычисление

> **Внимание:** В Python отрицательные числа с `~` дают неожиданный результат. Используйте `~mask & 0xFFFFFFFF` для 32-битного инвертирования.

3. Задание №15. Логика с отрезками (аналитический разбор)

Суть задания

Дано выражение: $(x \in A) \to ((x \in P) \lor (x \in Q))$ и отрезки P и Q. Нужно найти длину отрезка A (или минимальную/максимальную длину), при которой выражение истинно для **любого** x.

Метод: логические преобразования

Шаг 1. Избавляемся от импликации.

$$A \to B \equiv \lnot A \lor B$$

Наше выражение: $$(x \in A) \to ((x \in P) \lor (x \in Q))$$ Превращается в: $$\lnot(x \in A) \lor (x \in P) \lor (x \in Q)$$

Что означает: «x НЕ принадлежит A, ИЛИ x принадлежит P, ИЛИ x принадлежит Q».

Шаг 2. Анализируем, когда выражение может быть ЛОЖНО.

Выражение-ИЛИ ложно только если **все три части** ложны одновременно:

1. $\lnot(x \in A)$ ложно $\to$ $x \in A$ (x принадлежит A).
2. $(x \in P)$ ложно $\to$ $x \notin P$ (x НЕ принадлежит P).
3. $(x \in Q)$ ложно $\to$ $x \notin Q$ (x НЕ принадлежит Q).

То есть выражение ложно, когда x принадлежит A, но **не принадлежит ни P, ни Q**.

Шаг 3. Условие «истинно для любого x» означает:

Не должно существовать ни одного x, для которого выражение ложно. Следовательно, множество точек, одновременно принадлежащих A и НЕ принадлежащих $P \cup Q$, должно быть **пусто**.

$$A \cap \overline{(P \cup Q)} = \emptyset$$

Это значит: **A не должно выходить за пределы $P \cup Q$**.

$$A \subseteq (P \cup Q)$$

Шаг 4. Геометрический смысл.

A — это **один непрерывный отрезок**. $P \cup Q$ — это либо один отрезок (если P и Q пересекаются или примыкают), либо два отдельных отрезка (если между ними «дырка»).

Если $P \cup Q$ — один отрезок, то A может занимать его целиком. Если $P \cup Q$ — два отрезка с дыркой, то A не может «перепрыгнуть» дырку (A должен быть непрерывным). A может лежать либо целиком в P, либо целиком в Q.

Пример 1 (отрезки не пересекаются)

P = [10, 20], Q = [30, 40]. Расстояние между ними = 10.

$P \cup Q$ — два отдельных отрезка. A — непрерывный отрезок. A не может включать точки между 20 и 30 (они не в $P \cup Q$).

Значит, A может быть:

- Подмножеством P: максимальная длина = 20 − 10 = 10.
- Подмножеством Q: максимальная длина = 40 − 30 = 10.

Максимальная длина A = 10.

Пример 2 (отрезки пересекаются)

P = [10, 30], Q = [25, 40]. Пересечение: [25, 30].

$P \cup Q$ = [10, 40] — один непрерывный отрезок длиной 30.

Максимальная длина A = 30.

Пример 3 (отрезки примыкают)

P = [10, 20], Q = [20, 40]. Точка 20 принадлежит обоим.

$P \cup Q$ = [10, 40] — один отрезок длиной 30.

Максимальная длина A = 30.

Алгоритм для ЕГЭ

1. Запишите отрезки P и Q.
2. Найдите их объединение.
3. Если объединение — один непрерывный отрезок → длина A = длина объединения.
4. Если объединение — два отдельных отрезка → A = max(длина P, длина Q).

> **Более сложные случаи** (несколько переменных A, несколько отрезков) решаются аналогичным анализом: раскрываем импликации, находим условие ложности, требуем его невозможности.

4. Задание №18. Динамическое программирование в LibreOffice Calc

Суть задания

Дана прямоугольная таблица с числами. Исполнитель (Робот) движется из левой верхней клетки в правую нижнюю (или другую указанную). Двигаться можно только **вправо** и **вниз**. Нужно найти **максимальную** (или минимальную) сумму чисел, собранных по пути.

Метод: восходящее ДП прямо в ячейках Calc

Идея: В каждую клетку таблицы результатов записываем лучшую сумму, с которой можно в эту клетку прийти.

Правило заполнения: `dp[i][j] = \text{число в клетке} + \max(dp[i-1][j], dp[i][j-1])`

Где `dp[i-1][j]` — клетка сверху, `dp[i][j-1]` — клетка слева.

Пошаговый алгоритм в LibreOffice

Пусть исходная таблица на **Листе 1** в диапазоне `A1:D10`.

Шаг 1. Создаём **Лист 2** для результатов ДП.

Шаг 2. В ячейку `A1` Листа 2 вводим:

```
calc
=Лист1.A1
```

(Начальная клетка — просто копируем исходное значение.)

Шаг 3. Заполняем первую строку (можно прийти только **слева**). В `B1`:

```
calc
=A1 + Лист1.B1
```

Протягиваем формулу вправо на всю первую строку.

Шаг 4. Заполняем первый столбец (можно прийти только **сверху**). В `A2`:

```
calc
=A1 + Лист1.A2
```

Протягиваем формулу вниз на весь первый столбец.

Шаг 5. Заполняем внутренние клетки. В `B2`:

```
calc
=МАКС(A2; B1) + Лист1.B2
```

- `A2` — клетка сверху в таблице ДП.
- `B1` — клетка слева в таблице ДП.
- `Лист1.B2` — исходное число.

Протягиваем формулу на всю оставшуюся таблицу.

Шаг 6. Ответ — значение в правой нижней клетке таблицы ДП.

Конкретный пример

Исходная таблица (Лист 1):

	A	B	C
1	1	2	3
2	4	5	6
3	7	8	9

Таблица ДП (Лист 2):

	A	B	C
1	`=Лист1.A1` → 1	`=A1+Лист1.B1` → 3	`=B1+Лист1.C1` → 6
2	`=A1+Лист1.A2` → 5	`=МАКС(A2;B1)+Лист1.B2` → 10	`=МАКС(B2;C1)+Лист1.C2` → 16
3	`=A2+Лист1.A3` → 12	`=МАКС(A3;B2)+Лист1.B3` → 20	`=МАКС(B3;C2)+Лист1.C3` → 29

Ответ: 29 (ячейка C3).

Важные модификации

Задача	Модификация формулы
Минимальная сумма	`=МИН(сверху; слева) + значение`
Движение вверх и вправо	`=МАКС(снизу; слева) + значение`
Стены (клетки, куда нельзя)	Поставить `-1000000`
Количество путей	`=ЕСЛИ(сверху=слева; 2*сверху; МАКС(сверху;слева))` (зависит от задачи)

Типичная задача из ФИПИ

Таблица 20×20. Робот идёт из левого верхнего угла в правый нижний. Нужно найти максимальную сумму, избегая клеток с числом 0 (стены).

Решение:

1. В таблице ДП для клеток с 0 в исходной ставим очень маленькое число (например, `-1000000`).
2. Остальное — как в алгоритме выше.
3. Ответ — в правой нижней клетке.

Сводная таблица методов

Задание	Метод	Инструмент
№1	Двухшаговый визуальный	Бумага + карандаш
№13	Побитовые операции	Python
№15	Логические преобразования	Ручной анализ
№18	Восходящее ДП	LibreOffice Calc

Домашнее задание

Задание 1 (№1)

Возьмите любую таблицу расстояний из банка ФИПИ. Нарисуйте граф. Перерисуйте без пересечений. Приведите к виду схемы. Запишите ответ.

Задание 2 (№13)

Даны IP: `10.0.5.15` и `10.0.5.240`, маска `255.255.255.128`. Определите, в одной ли они сети. Найдите номер компьютера для второго адреса.

Задание 3 (№15)

P = [15, 35], Q = [25, 45]. Найдите наибольшую возможную длину A, при которой формула $(x \in A) \to ((x \in P) \lor (x \in Q))$ истинна для любого x.

Задание 4 (№18)

В LibreOffice решите задачу №18 из банка ФИПИ с таблицей 10×10 (движение вправо и вниз, максимизация суммы).

Что мы сегодня изучили

Задание	Ключевые моменты
№1	Рисование графа, удаление пересечений, визуальное совпадение
№13	`ip & mask`, `ip & ~mask`
№15	Преобразование импликации, анализ ложности, $A \subseteq P \cup Q$
№18	`=МАКС(сверху; слева) + значение`, протягивание формул

> **Связь с ЕГЭ:** Все четыре задания — по 1 баллу. Решаются руками или с минимальным Python-кодом.

Чек-лист занятия

- ☐ Умею рисовать граф по таблице и сопоставлять с условием визуально
- ☐ Знаю, как переводить IP и маску в числа и обратно на Python
- ☐ Могу анализировать логические выражения с отрезками
- ☐ Могу построить таблицу ДП в LibreOffice Calc для задачи №18
- ☐ Выполнено домашнее задание

Занятие 32. Кодирование, память, поиск (задания №4, 7, 10, 11)

План занятия

1. **Задание №4** — Кодирование, условие Фано, дерево кодов

2. **Задание №7** — Объём памяти (изображения, звук, текст)
3. **Задание №10** — Текстовый поиск
4. **Задание №11** — Информационный объём, пароли

> **Цель занятия:** Освоить четыре «ручных» задания ЕГЭ. Это 4 первичных балла.

1. Задание №4. Кодирование, условие Фано

Суть задания

Нужно закодировать символы (буквы, цифры) двоичными кодами так, чтобы выполнялось **условие Фано**: никакой код не является началом другого кода. Это гарантирует однозначное декодирование.

Основные понятия

Префиксный код — код, в котором никакое кодовое слово не является **префиксом** (началом) другого кодового слова.

Пример префиксного кода:

```
А — 0
Б — 10
В — 110
Г — 111
```

Проверим:

- `0` не является началом `10`, `110`, `111`.
- `10` не является началом `110`? Является! `110` начинается с `10`. **Это нарушение!**

Исправим:

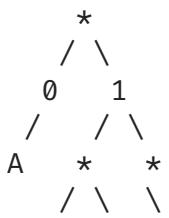
```
А — 0
Б — 10
В — 111
Г — 110
```

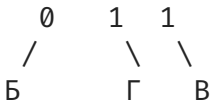
Теперь `10` не начало `111` и `110`? `110` начинается с `11`, а не с `10`. Всё хорошо.

Дерево кодов

Префиксный код можно представить в виде **двоичного дерева**:

- Листья — кодируемые символы.
- Путь от корня до листа — код символа (левый поворот = 0, правый = 1).





Здесь:

- А: 0
- Б: 100
- Г: 101
- В: 11

Типовые вопросы

1. **Даны коды некоторых букв, найти минимальную сумму длин кодов для остальных.**
2. **Дана строка, закодированная неравномерным кодом. Однозначно ли декодируется?**
3. **Какой наименьшей длины может быть код для буквы X?**

Метод решения

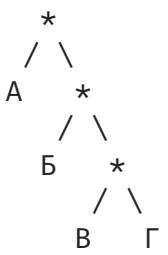
Строим дерево:

1. Берём известные коды и размещаем их в дереве.
2. Свободные листья — места для новых букв.
3. Выбираем свободные листья так, чтобы минимизировать сумму длин кодов (для частых букв — короткие коды).

Пример:

Дано: А — 0, Б — 10. Нужно закодировать В, Г, Д с минимальной суммой длин.

Дерево:



Свободные листья: 110, 111. Выбираем для В и Г. Д = 110, Г = 111. Сумма длин: 1 + 2 + 3 + 3 = 9.

Важно

- Условие Фано — **достаточное**, но не **необходимое** для однозначного декодирования.
- Иногда в задачах просят «выполняется ли условие Фано?» — нужно проверить все пары кодов.
- Для ЕГЭ достаточно знать определение и уметь строить дерево.

2. Задание №7. Объём памяти

Суть задания

Нужно вычислить объём памяти, занимаемый изображением, звуковым файлом или текстом. Все формулы сводятся к одной идее:

$$\text{Объём} = \text{Количество элементов} \times \text{Бит на элемент}$$

Изображения

Формула:

$$V = W \times H \times b$$

Где:

- W — ширина в пикселях.
- H — высота в пикселях.
- b — бит на пиксель (глубина цвета).

Если есть палитра из N цветов: $b = \lceil\log_2 N\rceil$ (округляем вверх).

Пример: Изображение 800×600, 256 цветов.

- $b = \log_2 256 = 8$ бит на пиксель.
- $V = 800 \times 600 \times 8 = 3\,840\,000$ бит = $480\,000$ байт ≈ 468.75 КБ.

Звук

Формула:

$$V = F \times R \times T \times C$$

Где:

- F — частота дискретизации (Гц) — сколько измерений в секунду.
- R — разрешение (бит) — сколько бит на одно измерение.
- T — время (секунды).
- C — количество каналов (1 — моно, 2 — стерео, 4 — квадро).

Пример: Стереозвук, 44.1 кГц, 16 бит, 60 секунд.

- $V = 44\,100 \times 16 \times 60 \times 2 = 84\,672\,000$ бит ≈ 10.1 МБ.

Текст

Формула:

$$V = N \times i$$

Где:

- N — количество символов.
- i — бит на символ (информационный вес символа).

Если алфавит из M символов: $i = \lceil\log_2 M\rceil$.

Перевод единиц

```
1 байт = 8 бит
1 КБ = 1024 байт
1 МБ = 1024 КБ
1 ГБ = 1024 МБ
```

> **Внимание:** На ЕГЭ могут попросить ответ в КБ или МБ. Обязательно проверяйте, в каких единицах нужен ответ.

3. Задание №10. Текстовый поиск

Суть задания

Дан текстовый файл. Нужно найти количество вхождений определённого слова или фразы. Решается **без программирования** — с помощью встроенного поиска в текстовом редакторе.

Инструменты

Встроенный текстовый редактор ЕГЭ (Блокнот / Notepad):

- `Ctrl+F` — открыть окно поиска.
- Ввести искомое слово.
- Кнопка «Найти далее» или «Подсчитать». В некоторых версиях количество вхождений показывается сразу.

LibreOffice Writer:

- `Ctrl+F` → ввести слово → «Найти всё». **VS Code / Notepad++:**
- `Ctrl+F` → видно количество совпадений.
- Можно использовать регулярные выражения для сложных шаблонов.

Типичный вопрос

«Сколько раз в тексте встречается слово 'информатика'?»

Решение:

1. Открыть файл в Блокноте.
2. `Ctrl+F`.
3. Ввести «информатика».
4. Посчитать вхождения (нажимать «Найти далее» или посмотреть счётчик).

Нюансы

- **Регистр:** обычно не важен (ищут точное совпадение), но проверьте условие.
- **Слово целиком:** иногда нужно найти слово как отдельное слово, а не как часть другого. В продвинутых редакторах есть опция «Только слово целиком».
- **Пересечения:** например, в строке «AAA» слово «AA» встречается 2 раза (позиции 1 и 2).

4. Задание №11. Информационный объём, пароли

Суть задания

Нужно вычислить объём памяти для хранения паролей, номеров, кодовых слов и т.п. Всё сводится к формуле:

$$\text{Объём} = \text{Количество объектов} \times \text{Бит на объект}$$

Пошаговый алгоритм

1. Определяем мощность алфавита M — сколько разных символов можно использовать.

Пример: пароль из 10 символов: заглавные буквы (26), строчные (26), цифры (10). $M = 26 + 26 + 10 = 62$.

2. Находим информационный вес одного символа i:

$$i = \lceil\log_2 M\rceil$$

Где $\lceil \rceil$ — округление вверх до целого.

Для $M = 62$: $\log_2 62 \approx 5.95$, округляем до 6 бит.

3. Находим вес одного пароля I:

$$I = L \times i$$

Где L — длина пароля в символах.

4. Переводим в байты:

$$I_{\text{байт}} = \lceil I / 8 \rceil$$

Пароль хранится целым числом байт.

5. Умножаем на количество пользователей:

$$V_{\text{общий}} = N \times I_{\text{байт}}$$

Где N — количество пользователей.

Пример

Пароль из 8 символов. Алфавит: 15 символов. 20 пользователей. Найти объём памяти.

1. $M = 15$.
2. $i = \lceil\log_2 15\rceil = \lceil 3.906\rceil = 4$ бит.
3. $I = 8 \times 4 = 32$ бит = $32 / 8 = 4$ байта.
4. $V = 20 \times 4 = 80$ байт.

Усложнённый вариант: дополнительная информация

Иногда для каждого пользователя хранится не только пароль, но и **дополнительные сведения** (ID, дата и т.п.). Тогда объём на пользователя = объём пароля + объём доп. сведений.

Пароли с ограничениями

Пример: Пароль состоит из 6 символов: первые два — буквы (26 заглавных), остальные — цифры (10).

$M_{\text{букв}} = 26$, $i_{\text{букв}} = \lceil\log_2 26\rceil = 5$ бит. $M_{\text{цифр}} = 10$, $i_{\text{цифр}} = \lceil\log_2 10\rceil = 4$ бит.

$I = 2 \times 5 + 4 \times 4 = 10 + 16 = 26$ бит. $I_{\text{байт}} = \lceil 26 / 8 \rceil = 4$ байта.

Шпаргалка по всем формулам

Задача	Формула
Бит на символ	$i = \lceil\log_2 M\rceil$
Объём пароля	$I = L \times i$ бит
В байты	$I / 8$, округлить вверх
Изображение	$V = W \times H \times b$
Звук	$V = F \times R \times T \times C$
Текст	$V = N \times i$
Количество цветов → бит	$b = \lceil\log_2 N_{\text{цветов}}\rceil$

Сводная таблица методов

Задание	Метод	Инструмент
№4	Построение дерева кодов	Ручной анализ
№7	Формулы объёма	Калькулятор
№10	Текстовый поиск	Ctrl+F
№11	Формулы информационного объёма	Калькулятор

Домашнее задание

Задание 1 (№4)

Даны коды: А — 0, Б — 10. Постройте префиксные коды минимальной длины для букв В, Г, Д, Е. Найдите сумму длин всех шести кодов.

Задание 2 (№7)

Изображение 1024×768 пикселей, 16 777 216 цветов. Сколько МБ занимает изображение?

Задание 3 (№10)

В любом текстовом файле найдите количество вхождений слова «и» (как отдельного слова).

Задание 4 (№11)

Пароль из 12 символов: буквы (12 строчных) и цифры (10). 100 пользователей. Дополнительно на каждого 10 байт служебной информации. Найдите общий объём в байтах.

Что мы сегодня изучили

Задание	Ключевые моменты
№4	Префиксный код, условие Фано, дерево кодов, минимизация суммы длин
№7	$V = W \times H \times b$, $V = F \times R \times T \times C$, $V = N \times i$
№10	Ctrl+F, подсчёт вхождений, регистр, целые слова
№11	$\lceil\log_2 M\rceil$, $I = L \times i$, округление до байт, доп. сведения

> **Связь с ЕГЭ:** Все четыре задания — по 1 баллу. Решаются руками или с помощью калькулятора.

Чек-лист занятия

- ☐ Понимаю условие Фано и умею строить дерево кодов
- ☐ Знаю формулы объёма для изображений, звука, текста
- ☐ Умею пользоваться поиском в текстовом редакторе
- ☐ Знаю, как вычислить информационный объём паролей
- ☐ Выполнено домашнее задание

Заключение

Занятие 33. Стратегия экзамена

План занятия

1. Общая структура экзамена
2. Порядок выполнения заданий
3. Распределение времени
4. Типичные ошибки и как их избежать
5. Чек-лист на экзамен

> **Цель занятия:** Собрать всё воедино. После этого занятия вы должны чётко знать, в каком порядке решать задачи, сколько времени на них тратить и как избежать обидных ошибок.

1. Общая структура экзамена

Основные цифры

- **Всего заданий:** 27.
- **Время:** 3 часа 55 минут (235 минут).
- **Максимальный первичный балл:** 29.
- **Минимальный порог:** 6 первичных баллов (40 тестовых).
- **Для 80+ тестовых:** нужно 20–22 первичных балла.
- **Для 100 тестовых:** нужно 27–29 первичных баллов.

Распределение заданий по инструментам

Инструмент	Задания	Баллы
Python	2, 5, 6, 8, 12, 14, 15, 16, 17, 19–23, 24, 25, 26, 27	19-20
Excel / LibreOffice	3, 9, 18	3
Ручной счёт	1, 4, 7, 10, 11, 13, 15 (отрезки)	6-7

2. Порядок выполнения заданий

Золотое правило

> **От простого к сложному. От быстрых баллов к трудозатратным.**

Рекомендуемый порядок

Этап 1. «Быстрые баллы» — ручные задачи (≈30 минут)

№	Задание	Время	Инструмент
1	Графы	2 мин	Визуальный анализ
4	Кодирование	3 мин	Дерево кодов
7	Объём памяти	3 мин	Формулы
10	Текстовый поиск	2 мин	Ctrl+F
11	Информационный объём	3 мин	Формулы
13	Сети и маски	5 мин	Python или ручной расчёт
15	Логика (отрезки)	5 мин	Аналитические преобразования

Этап 2. Excel / LibreOffice (≈15 минут)

№	Задание	Время	Инструмент
3	Базы данных	7 мин	ВПР, фильтрация
9	Электронные таблицы	5 мин	СЧЁТЕСЛИ, СУММЕСЛИ
18	ДП в таблице	5 мин	Протягивание формул

Этап 3. Python — брутфорс (≈30 минут)

№	Задание	Время	Инструмент
2	Таблицы истинности	3 мин	`product` + `permutations`
5	Алгоритм с числами	5 мин	Функция + перебор N
6	Анализ алгоритма	3 мин	Трассировка или Python
8	Комбинаторика (простые)	5 мин	`product`
14	Системы счисления	5 мин	Перебор x / основания
15	Логика (A — число)	5 мин	`all()` + перебор A

Этап 4. Python — рекурсия и ДП (≈50 минут)

№	Задание	Время	Инструмент
12	Редактор строк	5 мин	`while` + `replace`
16	Рекурсивные функции	5 мин	`@lru_cache`
17	Обработка файла	5 мин	Чтение + фильтрация
19–21	Теория игр	15 мин	`game()` + `any/all`
22	Параллельные процессы	5 мин	Рекурсия + кэш
23	Траектории	5 мин	`paths()` + `@lru_cache`
24	Строки из файла	5 мин	Скользящее окно

Этап 5. Python — сложные (≈40 минут)

№	Задание	Время	Инструмент
25	Делители, маски	10 мин	Перебор до √n
26	Сортировка, упаковка	15 мин	Жадный алгоритм
27	Пары/тройки, ДП	15 мин	ДП по остаткам

3. Распределение времени (3 часа 55 минут = 235 минут)

План

Этап	Задания	Плановое время	Нарастающий итог
1. Ручные задачи	1, 4, 7, 10, 11, 13, 15	25 мин	25 мин
2. Excel / LibreOffice	3, 9, 18	15 мин	40 мин
3. Python брутфорс	2, 5, 6, 8, 14, 15	30 мин	70 мин
4. Python рекурсия/строки	12, 16, 17, 19–21, 22, 23, 24	50 мин	120 мин
5. Python сложные	25, 26, 27	40 мин	160 мин
Резерв	Проверка, перенос в бланк	**75 мин**	235 мин

Почему такой большой резерв

- **Перенос ответов в бланк** — 10–15 минут.
- **Проверка** — перерешать 2–3 задачи, которые вызвали сомнения.
- **Зависшая задача** — если какая-то задача заняла больше времени, резерв всё компенсирует.
- **Нервы** — лучше иметь запас, чем паниковать в конце.

Что делать, если застрял на задаче

1. **Засеките 5 минут.** Если за это время нет прогресса — **пропустите**.
2. Напишите в черновике номер задачи и **вернитесь позже**.
3. **Не жертвуйте** лёгкими баллами ради одного сложного.

4. Типичные ошибки и как их избежать

Ошибка 1. Неправильный порядок выполнения

Начинают с №27, тратят час, не решают, паникуют, теряют лёгкие баллы.

Как избежать: Всегда от простого к сложному. Первые 10 заданий — за 30 минут.

Ошибка 2. Опечатка в формуле

Пропустили скобку, перепутали `and` / `or`, не тот остаток.

Как избежать: Проверять формулу на 1–2 примерах вручную. Использовать шаблоны из курса.

Ошибка 3. Забыли про `@lru_cache`

Рекурсивная функция работает бесконечно долго.

Как избежать: Всегда добавляйте `@lru_cache(maxsize=None)` к рекурсивным функциям.

Ошибка 4. Неправильный формат вывода

Вывели два числа через пробел, а нужно без пробела. Или наоборот.

Как избежать: Внимательно читайте условие. «Запишите подряд без разделителей» vs «запишите два числа».

Ошибка 5. Работа с файлами — не тот файл

Обрабатывают файл A вместо файла B, или забыли скачать файл.

Как избежать: Проверять имя файла в коде. Убедиться, что файл лежит в той же папке.

Ошибка 6. Не сохранили файл перед запуском

Среда зависла, код потерян.

Как избежать: `Ctrl+S` после каждой решённой задачи. Сохранять в облачную папку, если возможно.

Ошибка 7. Забыли про перенос ответов в бланк

Решили всё, но не перенесли ответы в бланк ответов.

Как избежать: Переносить ответы **сразу** после решения задачи. Не оставлять на конец.

Ошибка 8. Паника при ошибке

Программа выдаёт ошибку, время идёт, нарастает стресс.

Как избежать: Глубокий вдох. Ошибка — это нормально. Читайте сообщение об ошибке. 90% ошибок — опечатки.

5. Чек-лист на экзамен

Что взять с собой

- ☐ Паспорт.
- ☐ Чёрная гелевая ручка (2 шт).
- ☐ Вода в прозрачной бутылке.
- ☐

Шоколадка (быстрые углеводы для мозга).

Перед началом

- [ ] Проверить, что Python запускается.
- [ ] Проверить, что Excel / LibreOffice открывается.
- [ ] Проверить, что Блокнот работает.
- [ ] Открыть все файлы из задания, убедиться, что они читаются.
- [ ] Создать отдельную папку для своих программ.

Во время экзамена

- [ ] Решать в порядке: ручные → Excel → Python простые → Python сложные.
- [ ] После каждой задачи переносить ответ в бланк.
- [ ] `Ctrl+S` после каждой задачи.
- [ ] Если задача не идёт 5 минут — пропустить, вернуться позже.
- [ ] За 30 минут до конца: проверить все ответы, пересчитать сомнительные.
- [ ] За 15 минут до конца: финальная проверка бланка.

Технический чек-лист по Python

- [ ] Импортированы библиотеки: `itertools`, `functools`, `math`, `sys`.
- [ ] Для рекурсивных функций добавлен `@lru_cache(maxsize=None)`.
- [ ] Для больших рекурсий: `sys.setrecursionlimit(10000)`.
- [ ] Файлы открываются через `with open( ... ) as f:`.
- [ ] Ввод данных — через `int(input())` или чтение файла.

Чек-лист по заданиям

№	Статус	Ответ
1	☐	
2	☐	
3	☐	
4	☐	
5	☐	
6	☐	
7	☐	
8	☐	
9	☐	
10	☐	
11	☐	
12	☐	
13	☐	
14	☐	
15	☐	
16	☐	
17	☐	
18	☐	
19	☐	
20	☐	
21	☐	
22	☐	
23	☐	
24	☐	
25	☐	
26	☐	
27	☐	

Что мы изучили за весь курс

Раздел	Содержание	Задания
0. Введение	Структура ЕГЭ, инструменты	—
1. База Python	Синтаксис, типы, циклы, строки, списки, функции	2, 5, 6, 12, 17
2. Брутфорс	`product`, `permutations`, перебор N, x, A	2, 5, 8, 14, 15, 25
3. СС и строки	Перевод между СС, редактор строк, скользящее окно	5, 12, 14, 24
4. Рекурсия	Мемоизация, рекурсивный перебор, подсчёт путей, параллельные процессы	8, 16, 23, 22
5. Теория игр	`game()`, `any/all`, №19–21	19, 20, 21
6. ДП	Нисходящее/восходящее, цифровое ДП, ДП по остаткам	8, 16, 23, 27
7. Файлы и данные	Чтение файлов, сортировка, жадные алгоритмы	17, 26
8. Excel и ручные	Графы, сети, логика, ДП в таблице, кодирование, память, поиск, пароли	1, 3, 4, 7, 9, 10, 11, 13, 15, 18
9. Стратегия	Порядок, тайминг, ошибки, чек-лист	Все

Заключение

Вы прошли полный курс подготовки к ЕГЭ по информатике. У вас есть:

- **Шаблоны** для каждой задачи.
- **Методы** для каждого типа задач.
- **Стратегия** распределения времени.
- **Чек-лист** на экзамен.

Удачи на экзамене!

Финальный чек-лист

- ☐ Знаю порядок выполнения заданий
- ☐ Знаю, сколько времени тратить на каждый этап
- ☐ Знаю типичные ошибки и как их избежать
- ☐ Собрал вещи на экзамен
- ☐ Проверил работу программ и файлов
- ☐ Готов морально и физически