Практическое программирование

Python для робота

учебное пособие по изучению программирования
на примерах задач управления автономными объектами

Ю.Н.Поликарпов

ктн, снс

самостоятельное учебное электронное издание
предназначено для использования в учебном процессе
школьниками, студентами колледжей и ВУЗов,
обучающихся по физико-техническим специальностям,
а также всем интересующимся робототехникой и программированием

издание третье, переработанное и дополненное

СПб
Издатель Ю.Н.Поликарпов
2026

ISBN 978-5-6050573-3-8

УДК 372.862
ББК 74.263.2
П50

П50 **Поликарпов, Ю.Н.** Практическое программирование. Python для робота : учебное пособие по изучению программирования на примерах задач управления автономными объектами : самостоятельное учебное электронное издание / Ю.Н.Поликарпов. — 3-е изд., перераб. и доп. — СПб. : издатель Ю.Н.Поликарпов, 2026. — 1 CD-R (8 Мб). — ISBN 978-5-6050573-3-8. — Текст : электронный.

Пособие посвящено практическому программированию с использованием недорогого школьного робота, IDE Olly, двух программных симуляторов. Схема сборки робота приведена в приложении. В пособии имеется большое количество практических заданий, способствующих лучшему усвоению материала и развитию алгоритмического мышления.

В первую очередь пособие адресовано школьникам и студентам колледжей и ВУЗов, обучающимся по физико-техническим специальностям, но будет полезно всем интересующимся робототехникой и программированием.

В издании использованы материалы НИОКТР “Ветеран”, рег. №123033000025-9 (приложения Б, В).

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 01.07.2026.
Объем издания 8 МБ.
Комплектация 1 CD-R, пластиковый бокс.
Тираж 100 экз.
Издатель Ю.Н.Поликапов, newwave50@bk.ru, www.dec.su, +79516739548.

Оглавление

Техника безопасности

Прежде чем непосредственно перейти к содержанию книги необходимо сказать несколько слов о соблюдении мер предосторожности при работе.

Для изучения материала данного учебного пособия необходимо использовать электрическое оборудование - компьютер. Поэтому при работе следует соблюдать общие меры безопасности при работе с электричеством. Эти меры описаны во множестве источников информации, которые без труда можно найти на просторах интернета. Ниже перечислены основные из них.

Перед началом работы необходимо:

- убедиться в отсутствии видимых повреждений оборудования на рабочем месте;
- разместить на столе тетради, учебные пособия так, чтобы они не мешали работе на компьютере;
- принять правильною рабочую позу.

При работе не следует:

- работать в верхней или влажной одежде;
- класть одежду и посторонние предметы на рабочий стол;
- есть и пить за компьютером;
- присоединять или отсоединять кабели, трогать разъемы, провода и розетки при подключенном компьютере;
- при включенном питании передвигать компьютер и монитор;
- открывать системный блок;
- перекрывать вентиляционные отверстия на системном блоке и мониторе;
- ударять по клавиатуре, бесцельно нажимать на клавиши;
- класть книги, тетради и другие вещи на клавиатуру, монитор и системный блок;
- работать при плохом самочувствии.

Кроме того для сохранения своего здоровья во время работы за компьютером необходимо удерживать:

- расстояние от экрана до глаз – 70 – 80 см (расстояние вытянутой руки);
- вертикально прямую спину;

- плечи опущенными и расслабленными;
- ноги на полу и не скрещенными;
- локти, запястья и кисти рук на одном уровне;
- локтевые, тазобедренные, коленные, голеностопные суставы под прямым углом.

Введение

Данная книга является учебным пособием по программированию, помогающим изучению программирования на примере программирования робота и не является учебником по языку Python. При изучении программирования не так важен сам язык, как развитие алгоритмического мышления. Доказательством этого тезиса является книга “Искусство программирования” Д.Кнута, в которой в качестве языка вообще выбран язык гипотетической вычислительной машины.

Под алгоритмическим мышлением понимается умение планировать структуру действий, необходимых для достижения цели, при помощи фиксированного набора средств. Именно на развитии такого мышления следует делать акцент при изучении этой книги.

Важными компонентами алгоритмического мышления являются:

1. логический анализ задачи, в результате которого происходит разбиение большой задачи на малые;

2. анализ возможных состояний системы (робота) и реакций на них;

3. формальная запись алгоритма.

При изучении программирования развивается логическое мышление, умение видеть всю ситуацию целиком, совершенствуются навыки принятия решений.

При этом не важно на каком языке программирования происходит запись алгоритма. Тем не менее выбор языка Python не случаен. Во-первых, это современный алгоритмический язык высокого уровня. Во-вторых, с помощью этого языка достаточно удобно решается широкий круг задач, в том числе по робототехнике и искусственному интеллекту. В-третьих, это один из языков, используемых на ЕГЭ по информатике. В-четвертых, время разработки алгоритма на этом языке значительно меньше чем на C/C++, поэтому он хорошо подходит для исследовательских целей. После отработки алгоритма можно записать его на одном из более эффективных с точки зрения скорости выполнения языков, например C/C++.

Все задания в книге на уровне здравого смысла, упорство и осмысленность помогут выполнить все приведенные упражнения. Огромным подспорьем является то, что в книге приведены не чисто теоретические задания. Это практические задачи по программированию реального робота, собрать которого помогут приложения в конце книги. Кроме того, материал книги может быть проработан без робота — с использованием его программного симулятора. Наблюдение за поведением робота или его программной модели, анализ этого поведения помогут найти ошибки в написанном коде и легко их исправить.

Оптимальный возраст начала занятий по освоению программирования 10-11 лет (4-5 класс). В этом возрасте формируются навыки логического мышления, необходимые для работы по физико-техническим специальностям. Кроме того в этих классах происходит активное растаскивание школьников по спортивным секциям и

театрально-художественным кружкам, так что к моменту начала изучения физики и информатики свободных школьников, готовых много времени уделять изучению наукоемких предметов практически не остается. Впрочем, `никогда не поздно` встать на путь самурая, кто не сделал этого в 4-5 классе может сделать это и в более позднем возрасте, занятия программированием дадут полезный эффект в любом случае.

Наибольшую пользу начало обучения программированию с 4-5 классов может принести в школах с инженерными судо- и авиаклассами, организованными по поручению Президента России. Раннее обучение программированию позволит отобрать в инженерные классы наиболее склонных к такому обучению школьников. Хотя и в обычных школах такое обучение будет крайне полезно.

Глава 1

Знакомство с Python

1.1 Понятие программирования

Для управления роботом необходимо уметь давать ему различные команды, которые он может выполнять. Для этого существуют несколько методов. Например, метод дистанционного управления, когда команды подаются с помощью специального устройства - пульта дистанционного управления. Предметом этой книги является другой метод - программного управления, который подразумевает предварительное *программирование* робота - *оформление последовательности команд в программы*, с дальнейшей записью этих программ в память робота.

Сами программы пишутся на специальных алгоритмических языках. Эти языки служат для записи алгоритмов - набора команд, служащих для решения каких либо задач. Алгоритм записанный на алгоритмическом языке и есть программа. Одним из алгоритмических языков является Python. Преимуществом использования Python для программирования роботов является то, что для некоторых из них (точнее для некоторых микроконтроллеров, которые непосредственно управляют роботами) разработана специальная среда - MicroPython, которая работает на микроконтроллере, понимает язык Python и управляет роботами на основе программ, написанных на этом языке.

При этом программы могут быть очень большими и сложными. Для упрощения их разработки часто их делят на части, которые называются модулями и хранятся в отдельных файлах. Например, ниже представлен программный код модуля начальной загрузки `boot`, который хранится в файле boot.py, для робота, описанного в приложениях Б и В [1]:

Код 1.1: Код модуля начальной загрузки boot

```
# boot.py -- run on boot-up
# can run arbitrary Python, but best to keep it minimal

import machine
import pyb
from sys import path
path.append('/flash/inv')
pyb.country('US') # ISO 3166-1 Alpha-2 code, eg US, GB, DE, AU
pyb.main('/flash/main.py') # main script to run after this one
pyb.usb_mode('VCP') # act as a serial device
```

[1] В каталоге model компакт диска находится 3d модель робота, пригодная для печати на 3d принтере. При возникновении проблем с печатью можно обратиться к автору пособия для приобретения распечатанного комплекта пластиковых элементов робота.

```
11 #pyb.usb_mode('VCP+HID') # act as a serial device and a mouse
12 #import main
```

Модуль boot первым начинает выполняться при включении питания робота, работающего под управлением MicroPython. В этом модуле происходят некоторые необходимые настройки. Например, в 9 строке указывается, что после завершения его работы управление передается модулю `main`, находящемуся в файле main.py внутри каталога /flash.

Программирование робота можно выполнить с помощью интегрированной среды разработки Olly, которую можно скачать в интернете по адресу:

http://www.dec.su/upload/wysiwyg/olly-5.0.1-py3-none-any.whl.

Краткое описание работы с Olly приведено в следующем параграфе. Описание установки среды разработки Olly приведено в приложении Д.

1.2 Начало работы с интегрированной средой разработки (IDE) Olly

Если при первом запуске интегрированной среды разработки Olly не был выбран русский язык, она может иметь интерфейс на английском. Для включения русского языка необходимо в основном меню программы выбрать пункт “Tools” и в сплывающем меню выбрать пункт “Options...” (рисунок 1.1).

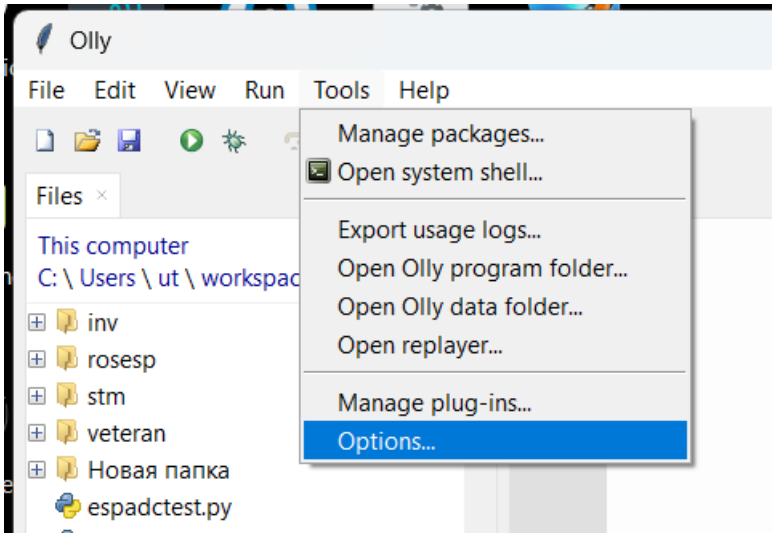


Рис. 1.1: Открытие окна настроек программы

В появившемся диалоговом окне на вкладке “General” в выпадающем списке “Language” следует выбрать “Русский язык” (рисунок 1.2).

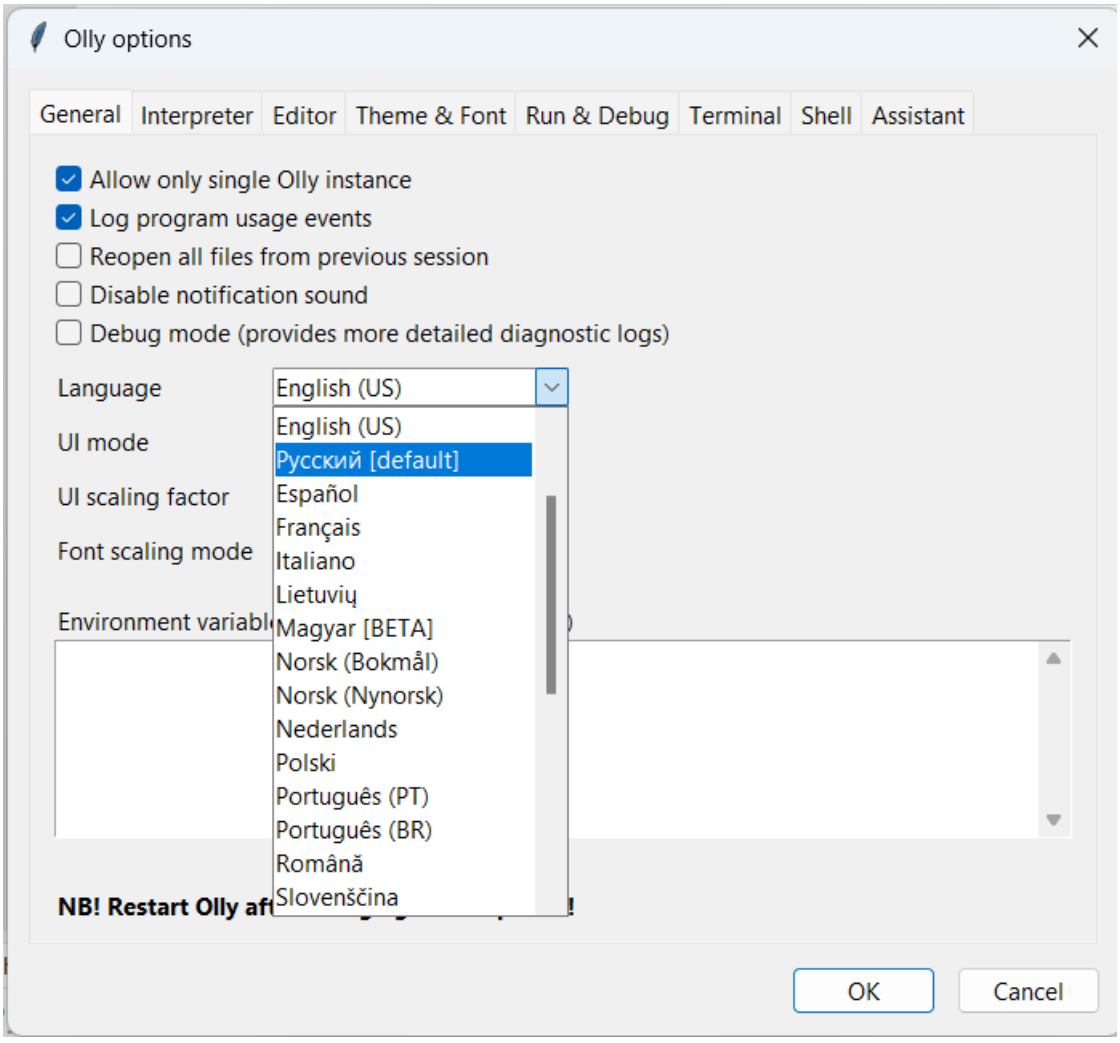


Рис. 1.2: Выбор русского языка для интерфейса программы

Далее следует нажать кнопку “OK” (рисунок 1.3), при следующем запуске интегрированной среды разработки Olly язык интерфейса изменится на русский (рисунок 1.4).

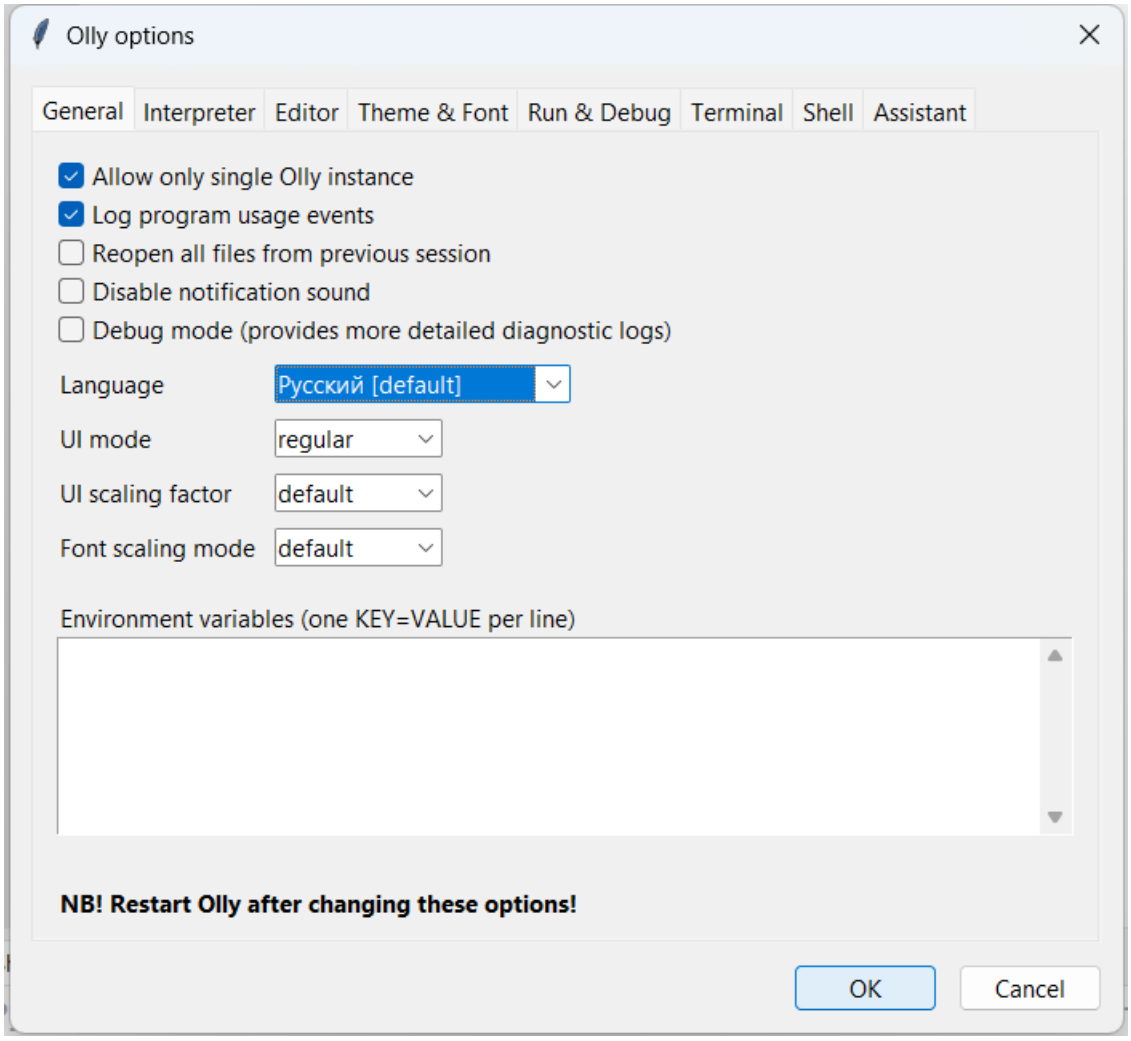


Рис. 1.3: Подтверждение выбора языка

Для удобства работы желательно, чтобы в левой части окна Olly была открыта панель проводника по файловой системе - вкладка “Файлы” (рисунок 1.4).

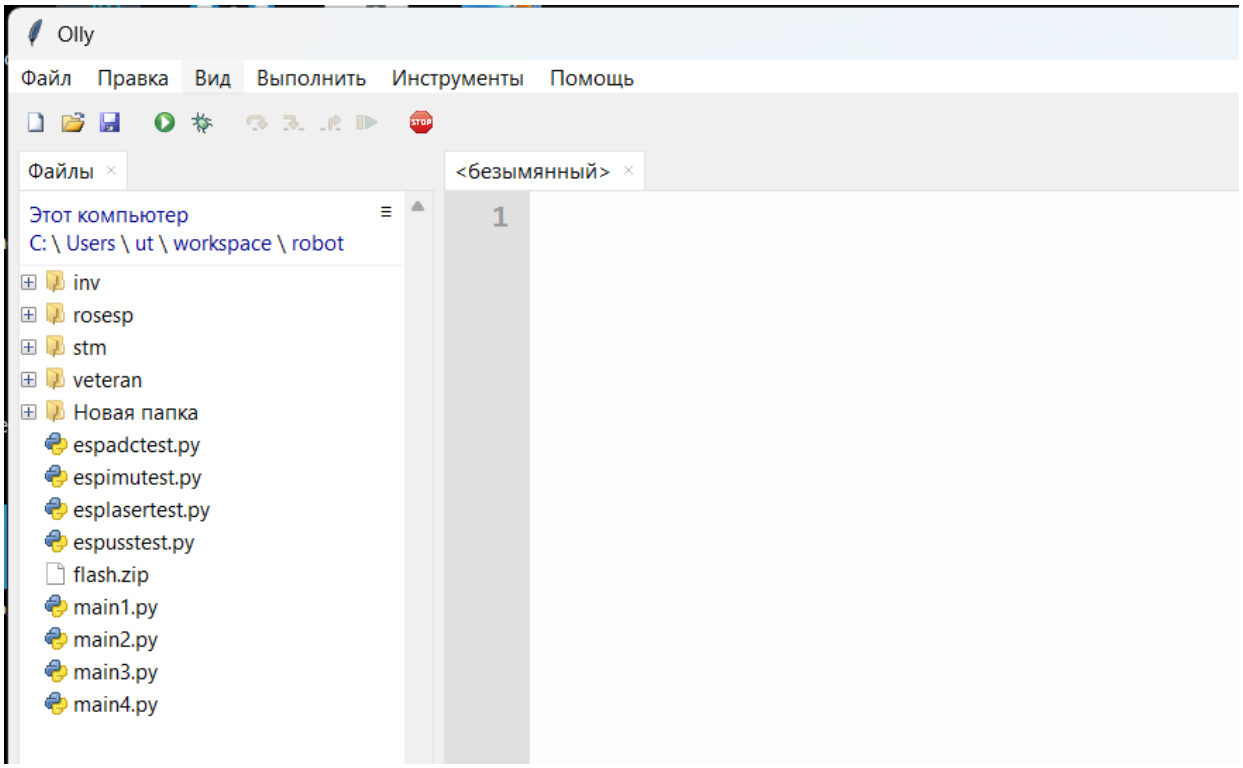


Рис. 1.4: Панель “Файлы” в левой части окна Olly

Чтобы эта панель была открыта необходимо в основном меню выбрать пункт “Вид” и в всплывающем меню поставить галочку напротив пункта “Файлы” (рисунок 1.5).

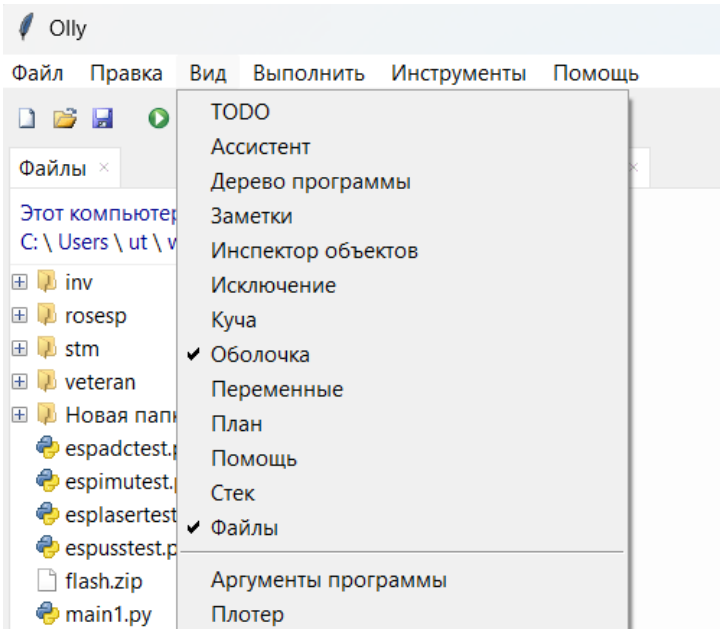


Рис. 1.5: Настройка видимости проводника по файловой системе

Использование интегрированной среды разработки Olly для программной симуляции робота

При изучении материала 1 и 2 главы для проверки работоспособности программного кода можно использовать один из симуляторов интегрированной среды разработки Olly (всего их два, конкретный симулятор определянтся именем модуля из которого производится импорт команд управления роботом, более простой симулятор больше подходит для глав 1 и 2, для последующих глав - более сложный, импортируемый из модуля nbase - описание этого симулятора находится в приложении **??**), хотя можно использовать и робота описанного в приложениях Б и В.

Например, для проверки программного кода 1.2:

Код 1.2: Некоторый программный код управления роботом

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

#rf(x) двигаться вперед со скоростью x: целое, 0<=x<=100
#rr(x) двигаться назад со скоростью x: целое, 0<=x<=100
#lt(x) разворачиваться влево, двигаясь по дуге радиусом x: целое,
    0<=x<=100
#rt(x) разворачиваться вправо, двигаясь по дуге радиусом x: целое,
    0<=x<=100
#st(x) сохранять текущий режим работы в течении x секунд: 0<=x
#stop() остановиться
#LED.value(0) зажечь светодиод
#LED.value(1) погасить светодиод
#LED.value() узнать не горит ли светодиод
#KEY.value() узнать не нажата ли кнопка

while KEY.value():
    pass

rf(90)
st(1.5)
lt(0)
st(2)
stop()
```

следует создать новый файл (комбинация клавиш “Ctrl”+“N”) и набрать в нем этот код (рисунок 1.6).

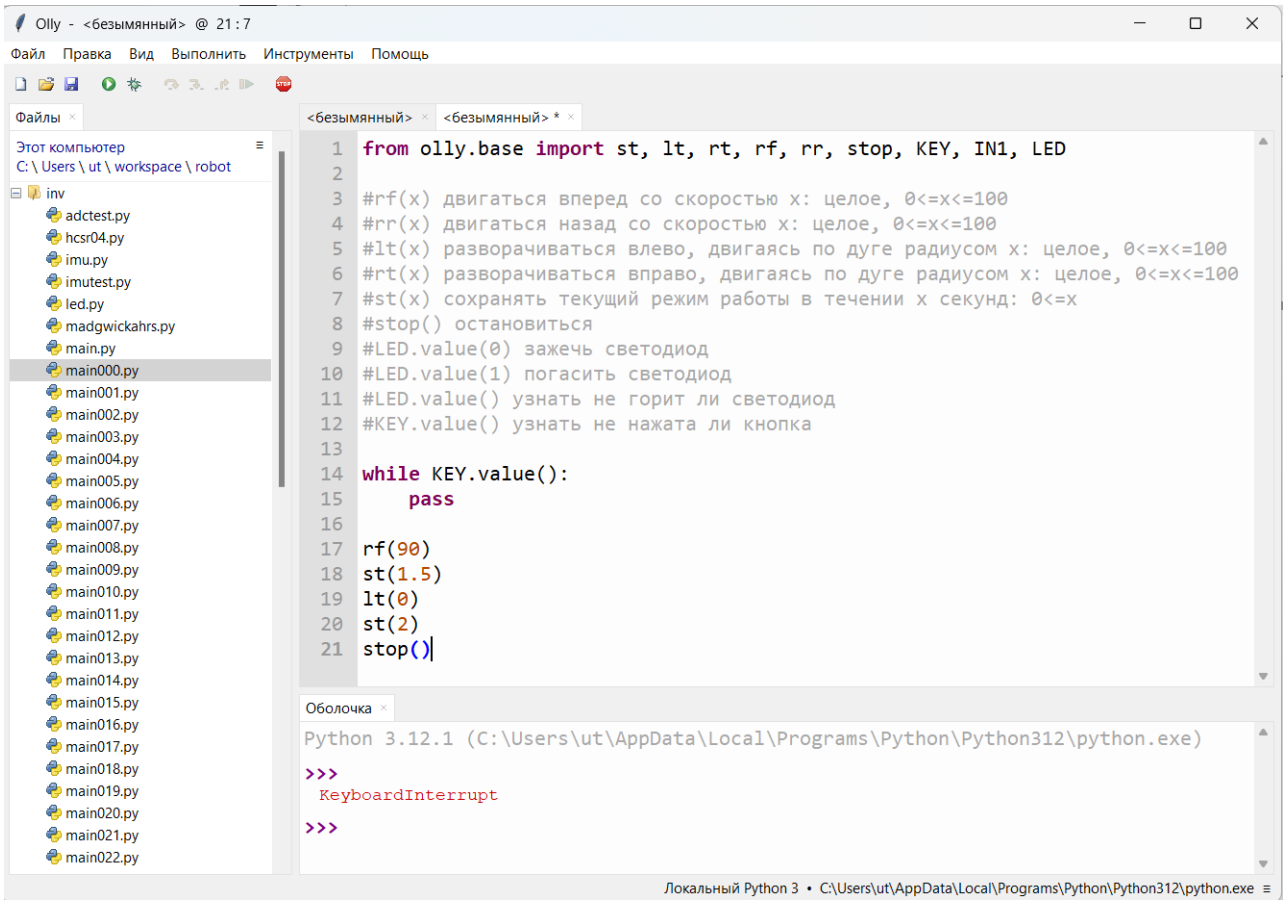


Рис. 1.6: Создание нового файла и набор в нем текста программного кода

После чего необходимо сохранить этот файл с программным кодом в каком-либо месте файловой системы компьютера. Удобно для этих целей создать папку со своим именем, например, в каталоге ws внутри папки Documents. Для этого необходимо правой кнопкой мыши нажать на папку ws (рисунок 1.7).

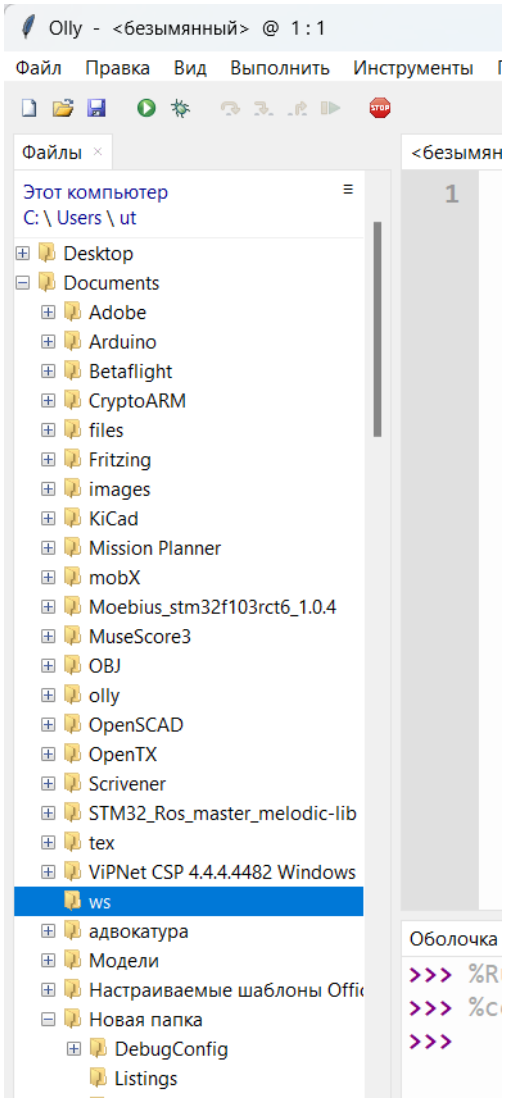


Рис. 1.7: Каталог ws внутри папки Documents

В появившемся всплывающем меню выбрать пункт “Новый каталог...” (рисунок 1.8).

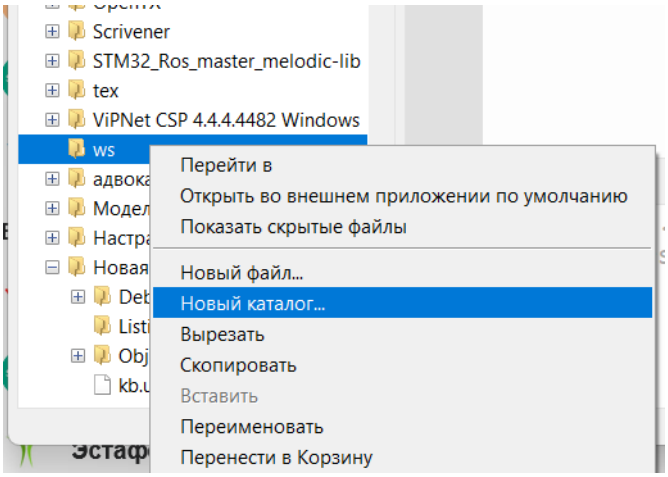


Рис. 1.8: Выбор пункта создания нового каталога

В открывшемся диалоговом окне указать, например, свою фамилию, после чего нажать кнопку “OK” (рисунок 1.9).

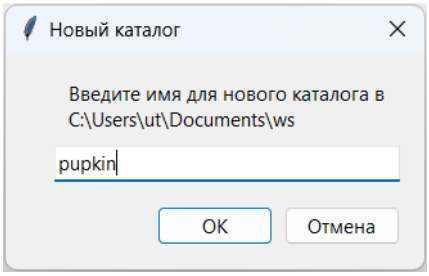


Рис. 1.9: Создание своего каталога

В результате в папке ws появится новый каталог (рисунок 1.10).

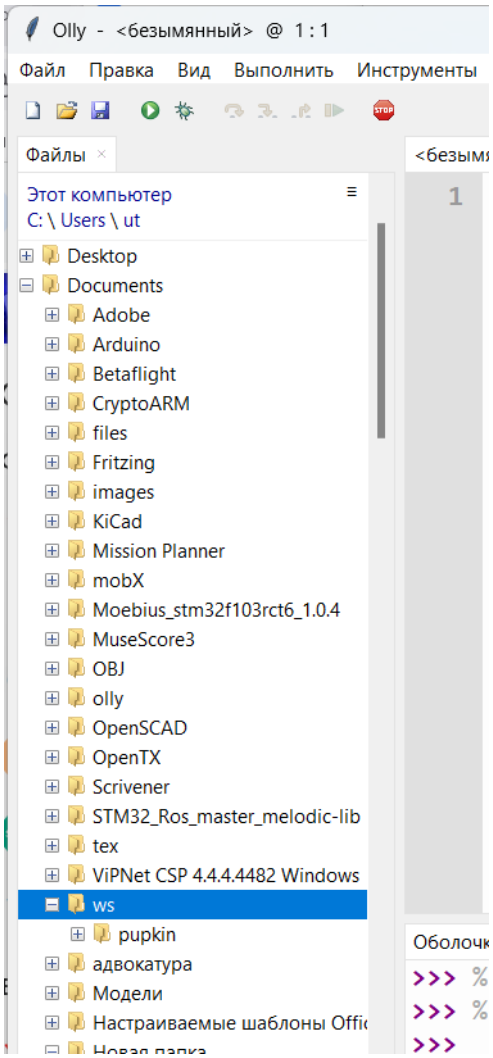


Рис. 1.10: Вновь созданный каталог

Теперь в этом каталоге можно сохранить созданный недавно программный файл, для чего необходимо нажать комбинацию клавиш “Ctrl”+“Shift”+“S” и в появившемся диалоговом окне выбрать место сохранения и имя файла (рисунок 1.11), после чего нажать кнопку “Сохранить”.

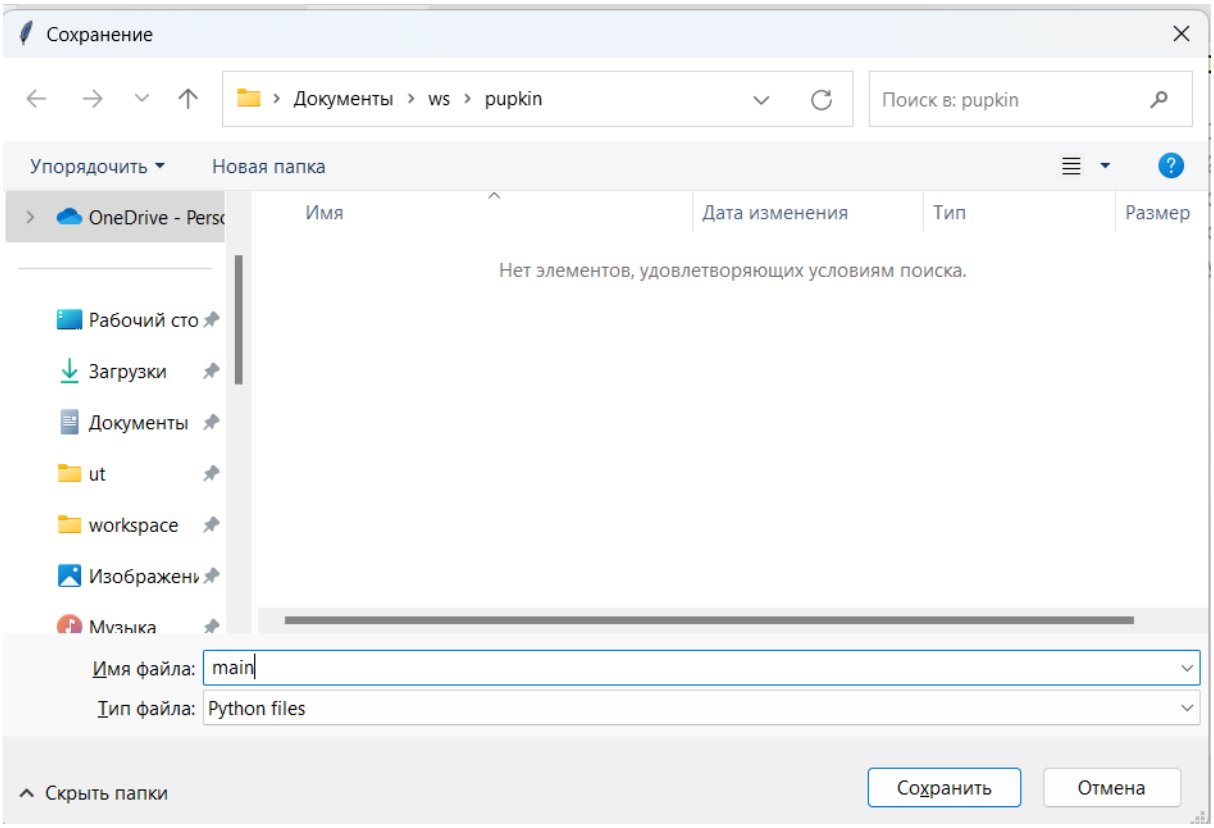


Рис. 1.11: Сохранение файла

После чего в файловой системе компьютера появится новый файл (рисунок 1.12).

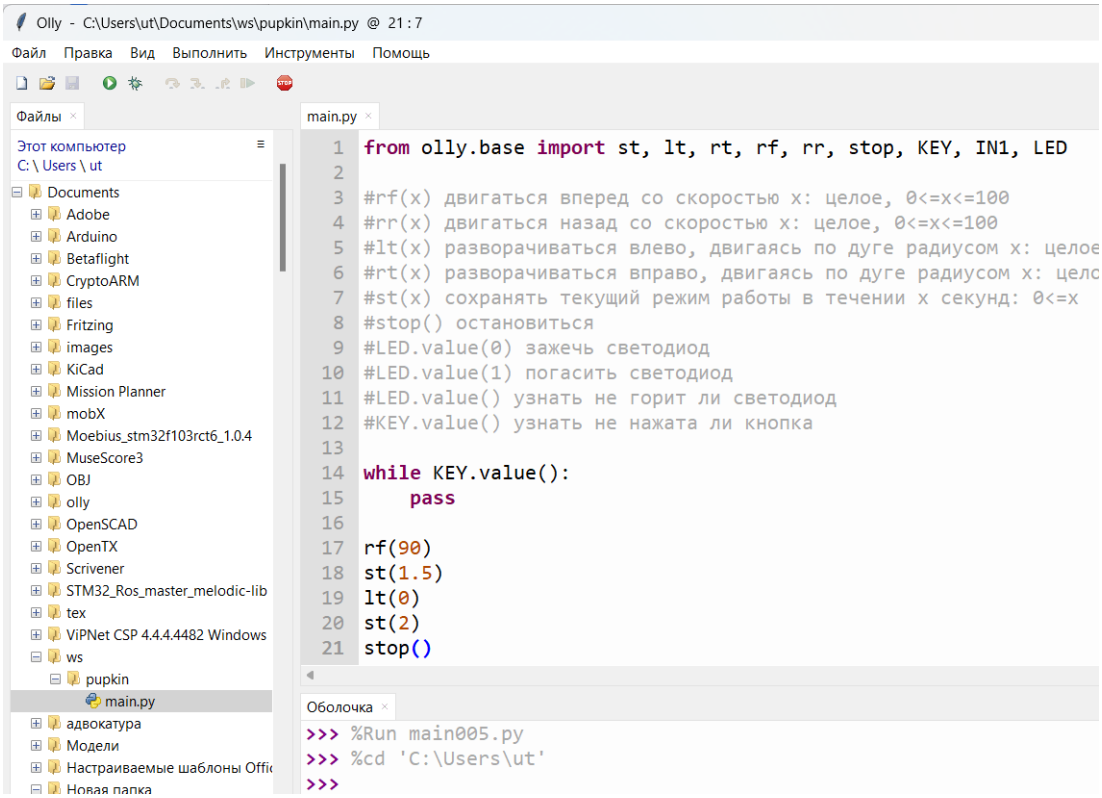


Рис. 1.12: Сохраненный файл

Затем следует убедиться, что установлена верная среда выполнения, для этого мышкой необходимо нажать кнопку в самом нижнем правом углу окна Olly (рисунок 1.13),

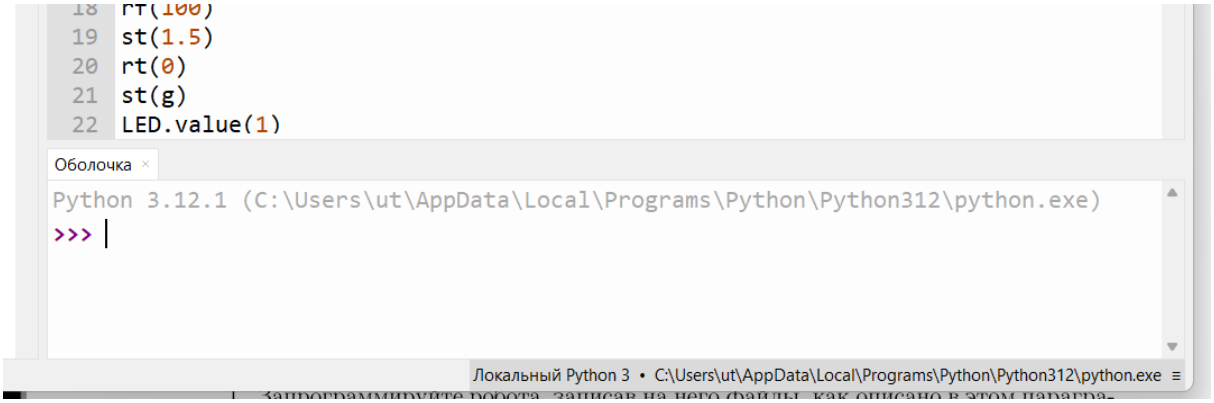


Рис. 1.13: Настройка среды выполнения

и в появившемся всплывающем меню выбрать строчку “Настроить интерпретатор...” (рисунок 1.14).

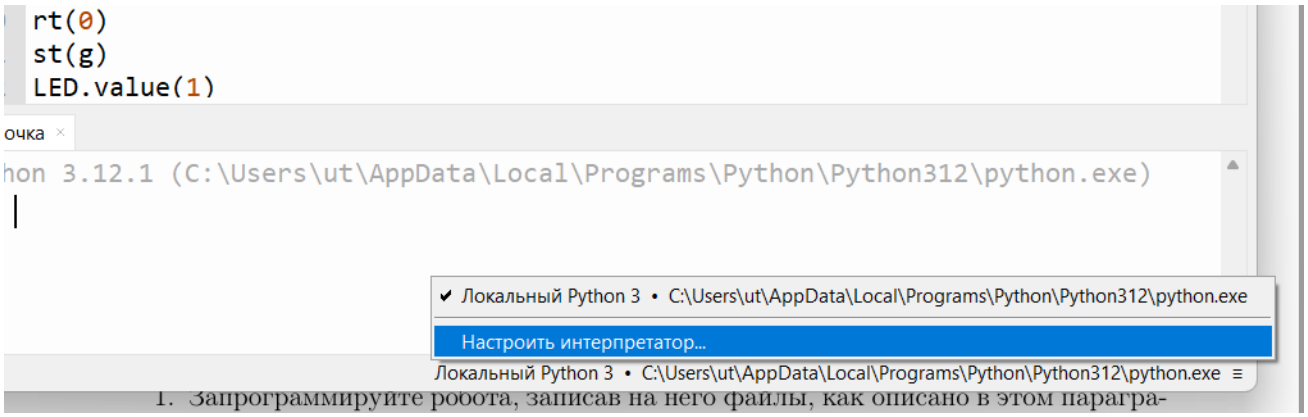


Рис. 1.14: Запуск настройки интерпретатора

Далее в появившемся диалоговом окне на вкладке “Интерпретатор” надо выбрать “Локальный Python3” (рисунок 1.15),

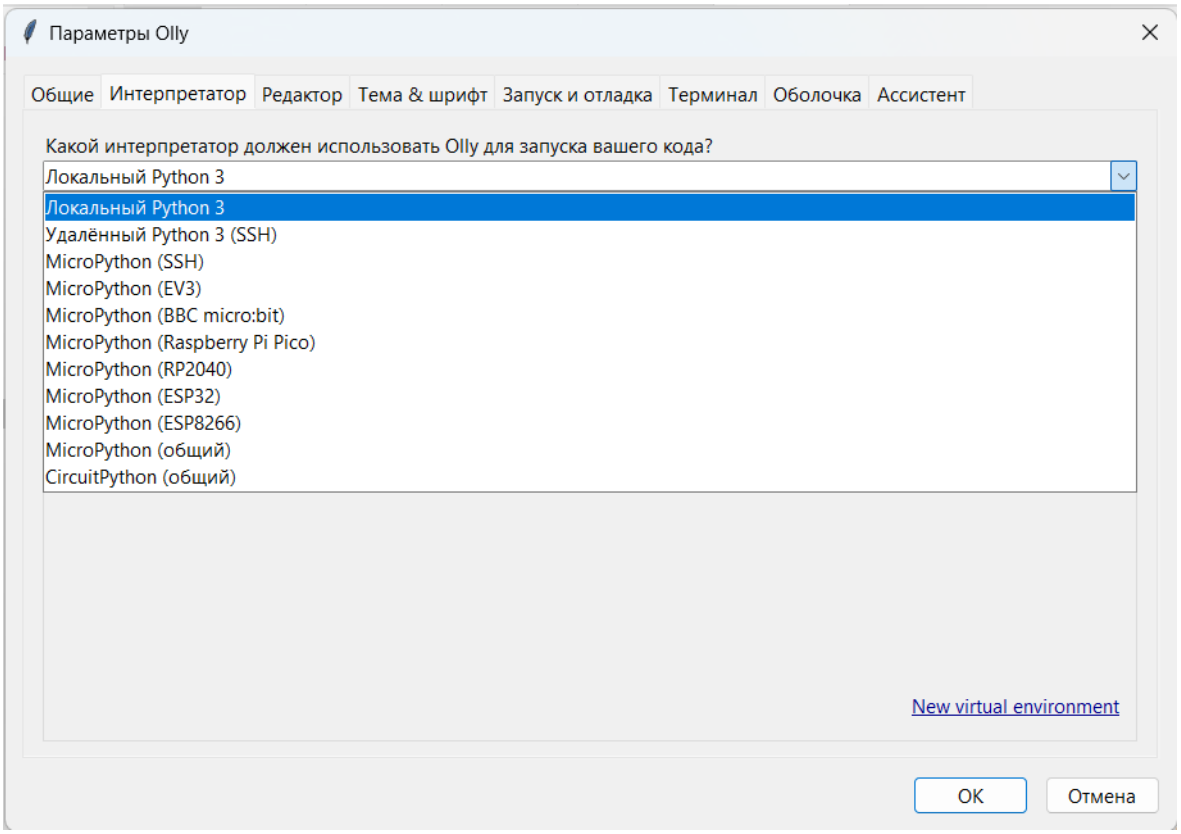


Рис. 1.15: Выбор среды выполнения

после чего следует нажать кнопку “OK” (рисунок 1.16).

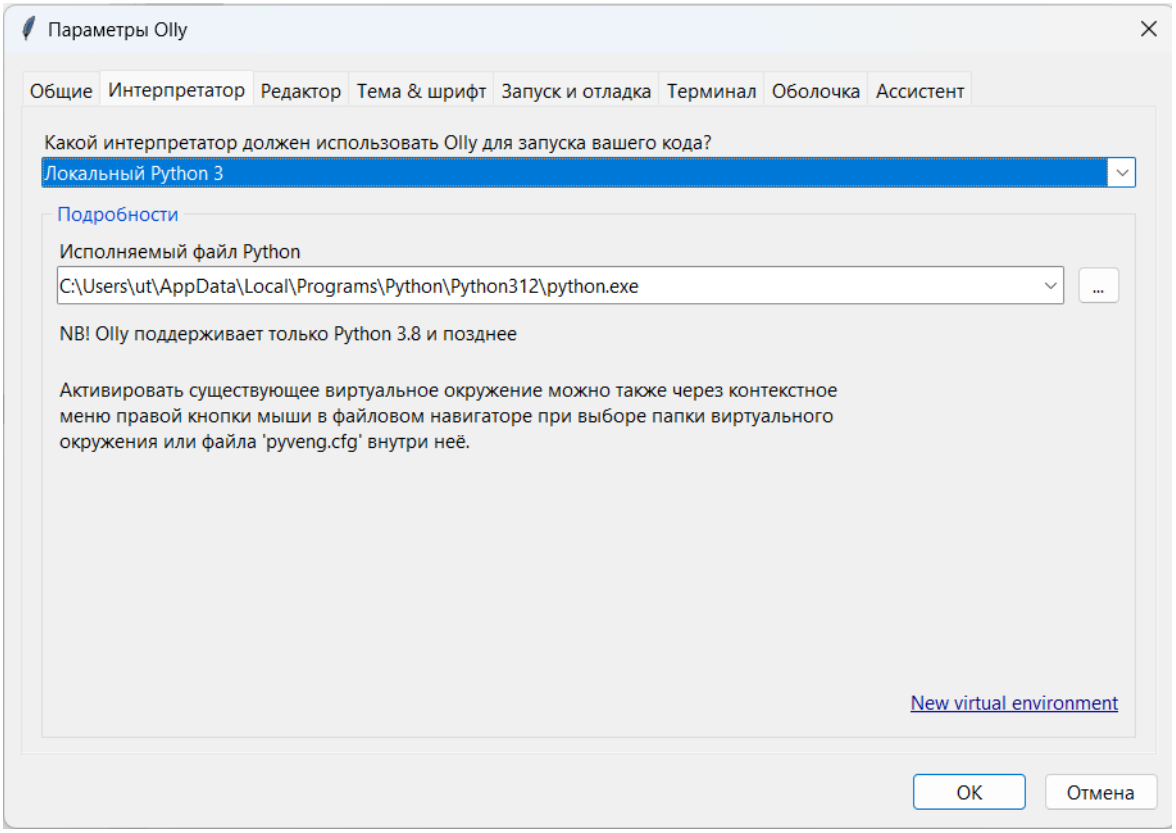


Рис. 1.16: Окончание настройки среды выполнения

Запуск сохраненного программного кода происходит при нажатии функциональной клавиши “F5”, при этом открывается графическое окно симулятора робота (рисунок 1.17) в котором робот изображен в виде стрелочки и стоит на месте ожидая нажатия кнопки (строки 13-14 кода 1.2).

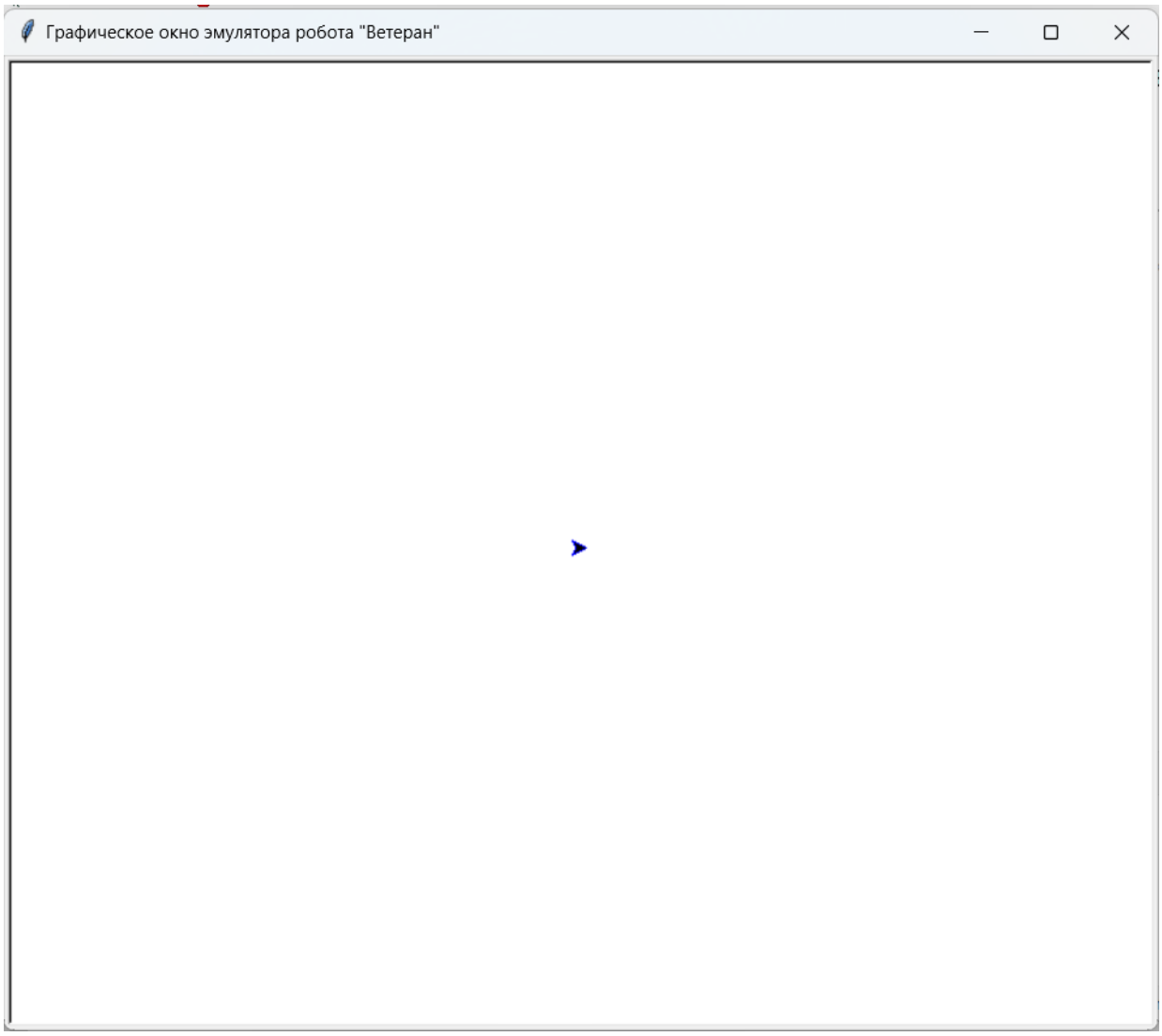


Рис. 1.17: Графическое окно симулятора робота

Симуляция нажатия кнопки на роботе осуществляется с помощью клавиши “Shift”, после нажатия которой робот начнет движение, при этом в графическом окне будет прочерчена траектория движения робота (рисунок 1.18). После окончания работы программы графическое окно симулятора следует закрыть нажатием на крестик в верхнем правом его углу.

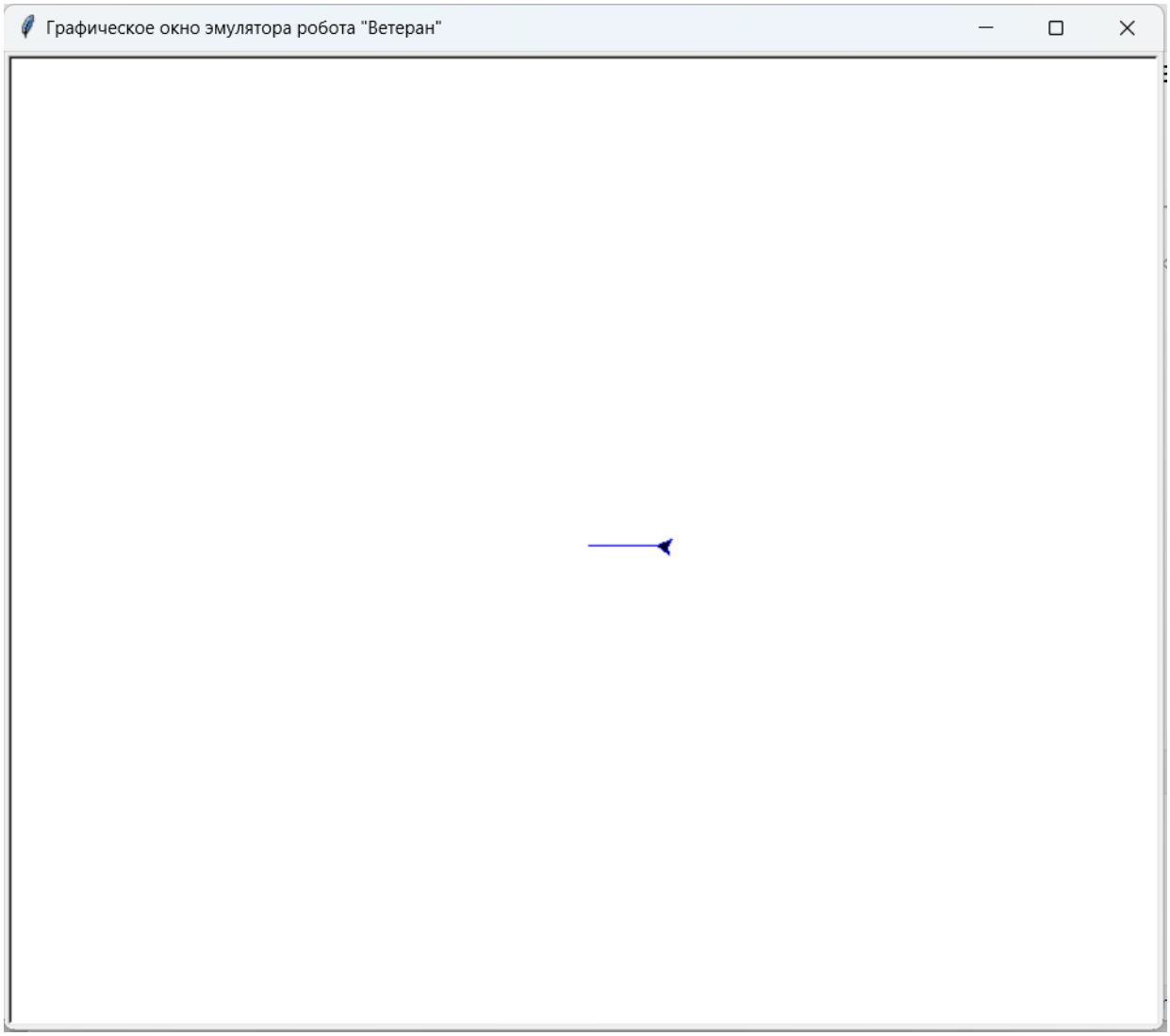


Рис. 1.18: Результат работы симулятора

Если в программе используются команды управления светодиодом, то в симуляторе работа светодиода робота будет отображаться изменением фона графического окна симулятора, кроме того соответствующие участки траектории робота будут изображены линиями разных цветов (рисунок 1.19).

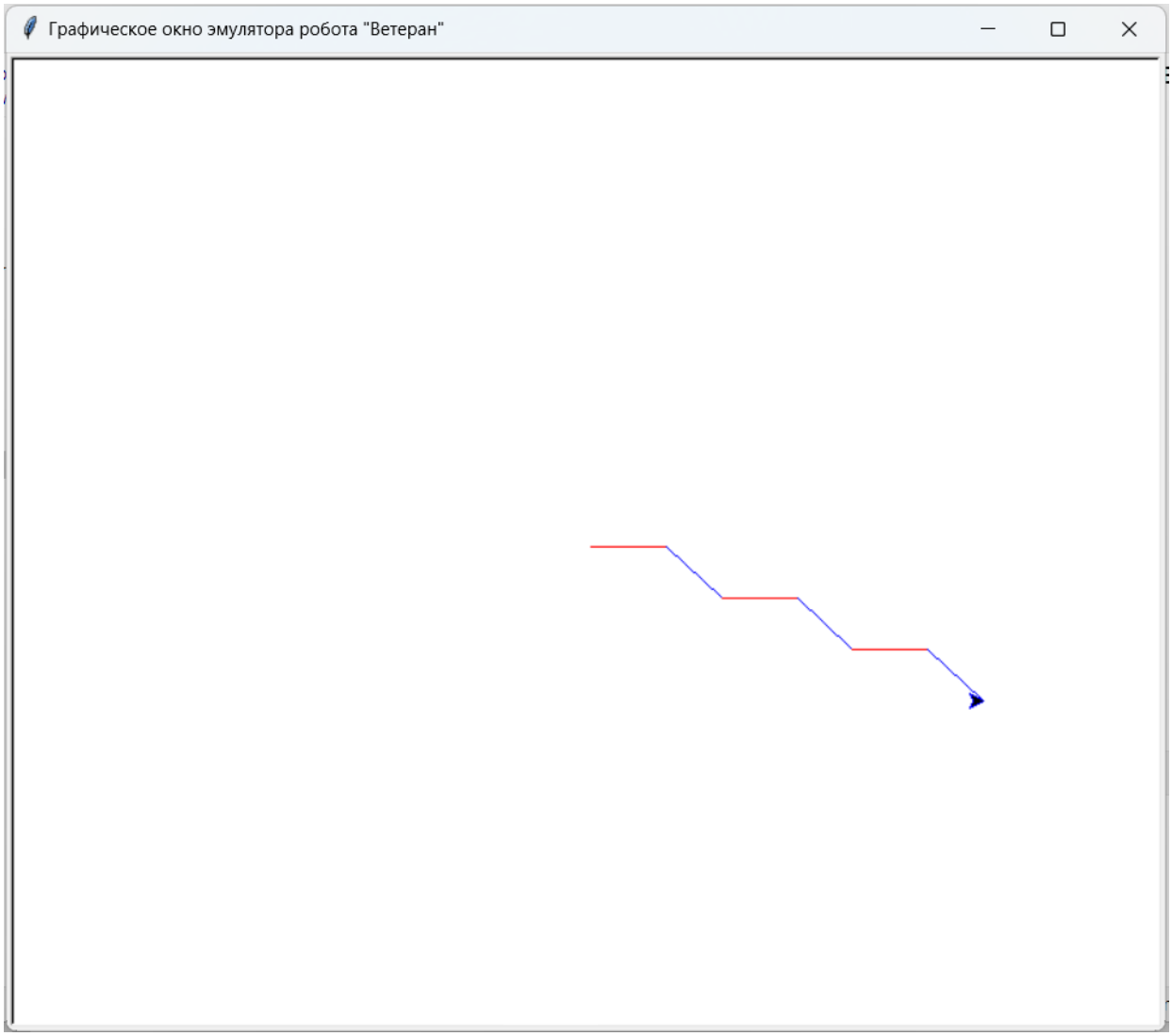


Рис. 1.19: Отображение работы светодиода в симуляторе

Использование интегрированной среды разработки Olly для программирования робота “Ветеран”

При выполнении заданий 3 и 4 глав необходим физический робот, описанный в приложениях Б и В, его так же можно использовать в главах 1 и 2.

Подключение робота к компьютеру

Для программирования робота с помощью интегрированной среды разработки Olly необходимо установить USB соединение робота с компьютером с помощью USB кабеля. Для организации дальнейшего взаимодействия робота и компьютера необходимо выполнить некоторые настройки в среде Olly. В частности в качестве среды выполнения необходимо выбрать MicroPython, прошитый на роботе, для этого мышкой необходимо нажать кнопку в самом нижнем правом углу окна Olly (рисунок 1.20),

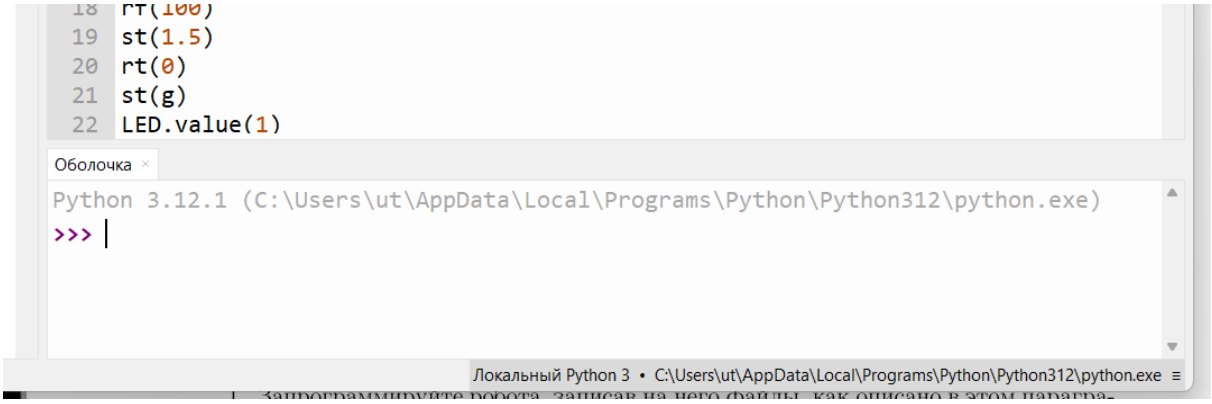


Рис. 1.20: Настройка среды выполнения

и в появившемся всплывающем меню выбрать строчку “Настроить интерпретатор...” (рисунок 1.21).

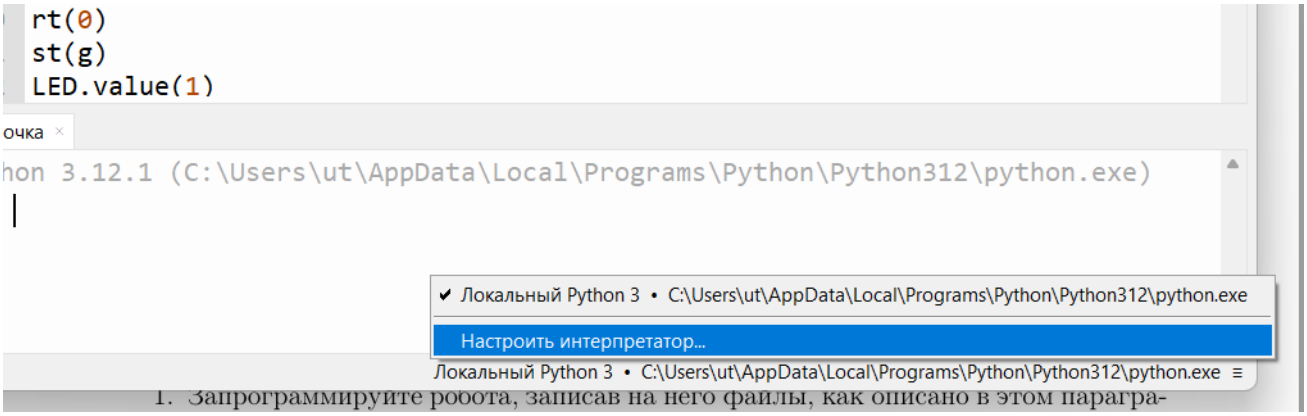


Рис. 1.21: Запуск настройки интерпретатора

Далее в выпадающем списке появившегося диалогового окна на вкладке “Интерпретатор” надо выбрать “MicroPython (общий)” (рисунок 1.22),

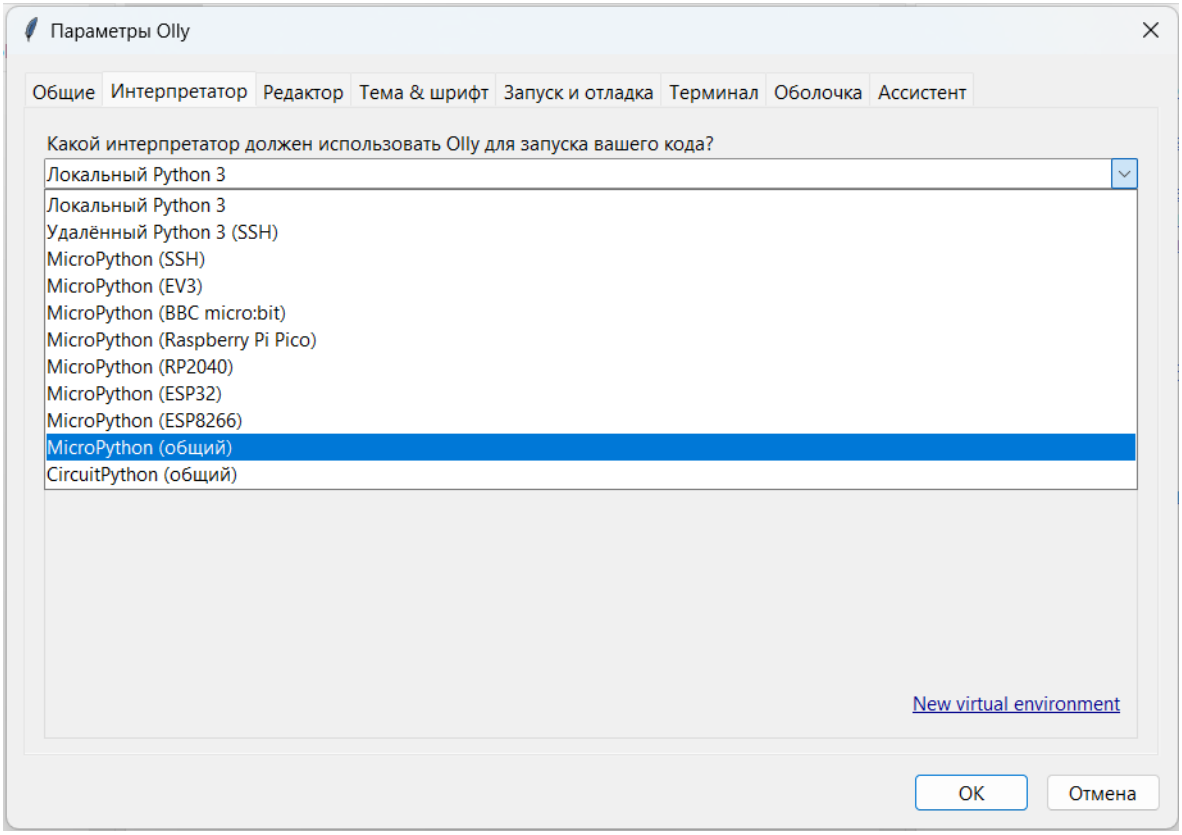


Рис. 1.22: Выбор среды выполнения

Затем в выпадающем списке Порт выбрать строчку содержащую “USB@COMx”, где x-номер COM порта робота (рисунок 1.23).

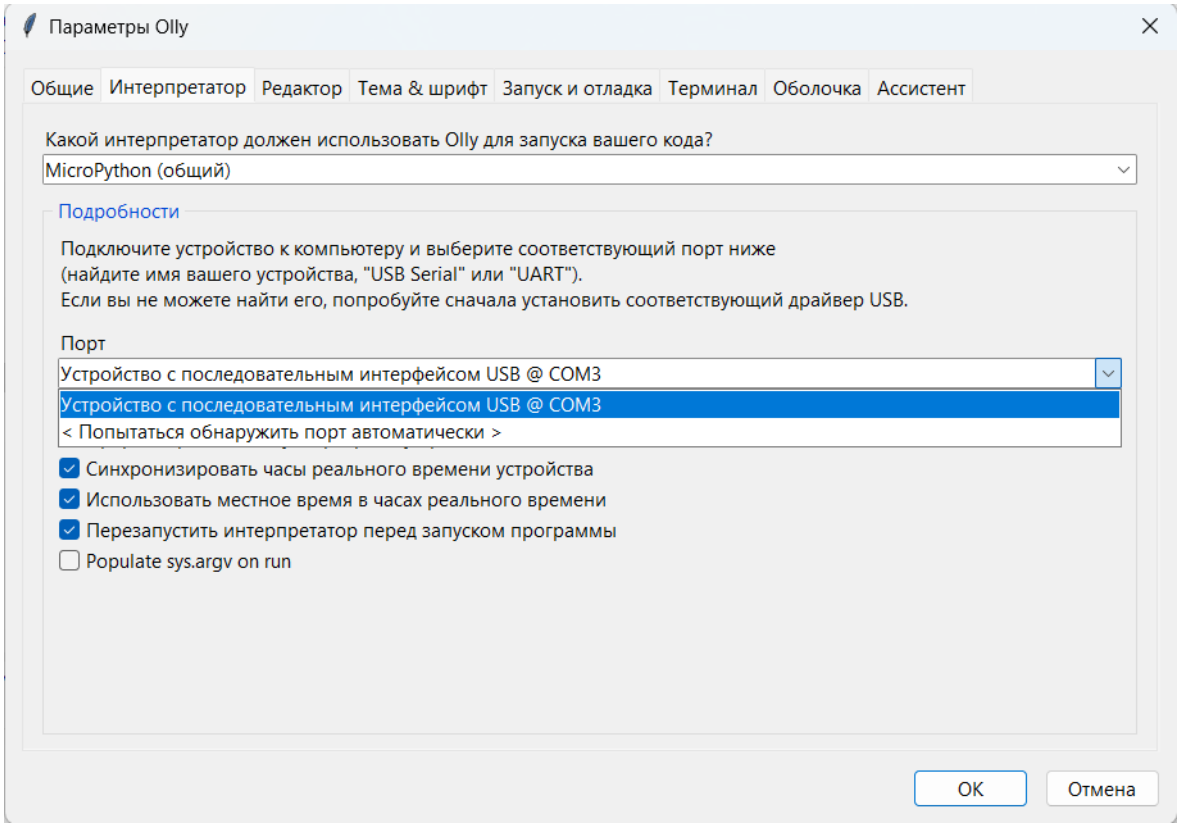


Рис. 1.23: Выбор COM порта робота

После чего следует подтвердить настройки нажав кнопку “OK” (рисунок 1.24).

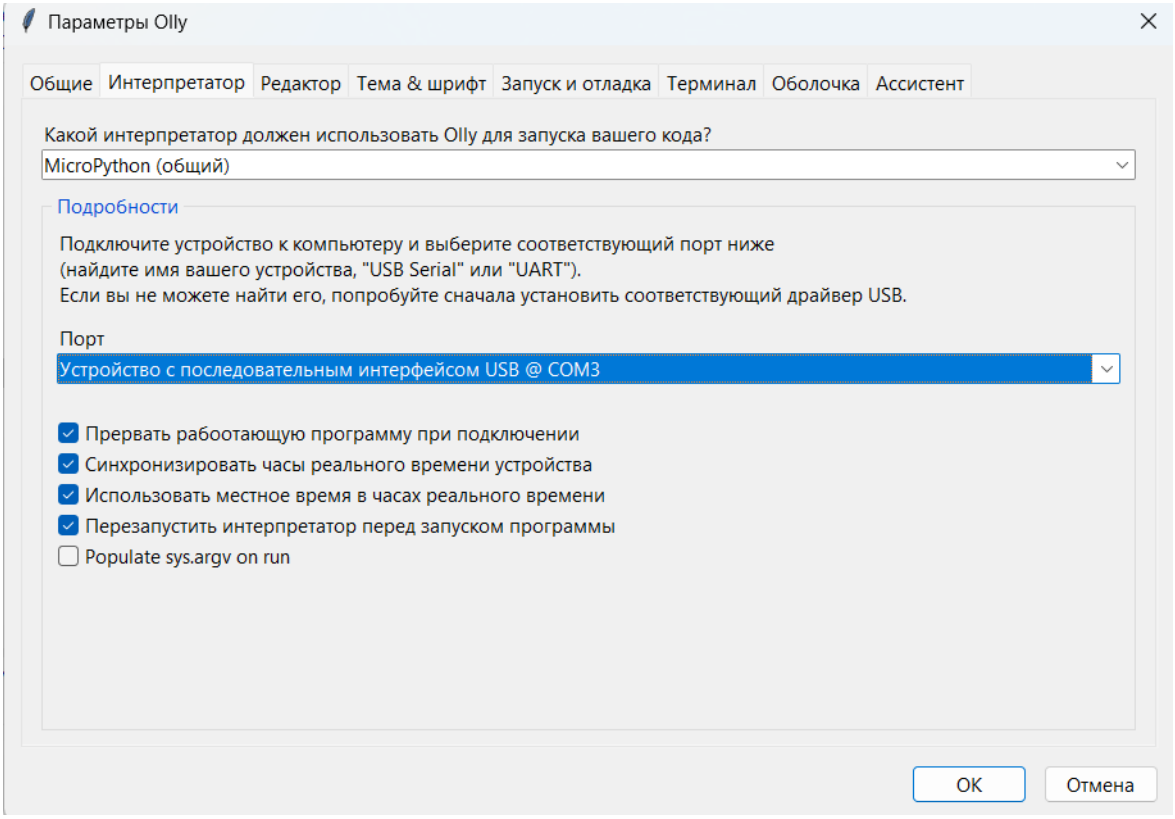


Рис. 1.24: Подтверждение настроек

Далее для синхронизации работы интегрированной среды разработки и робота следует нажать красную кнопку “STOP” в IDE Olly (рисунок 1.25).

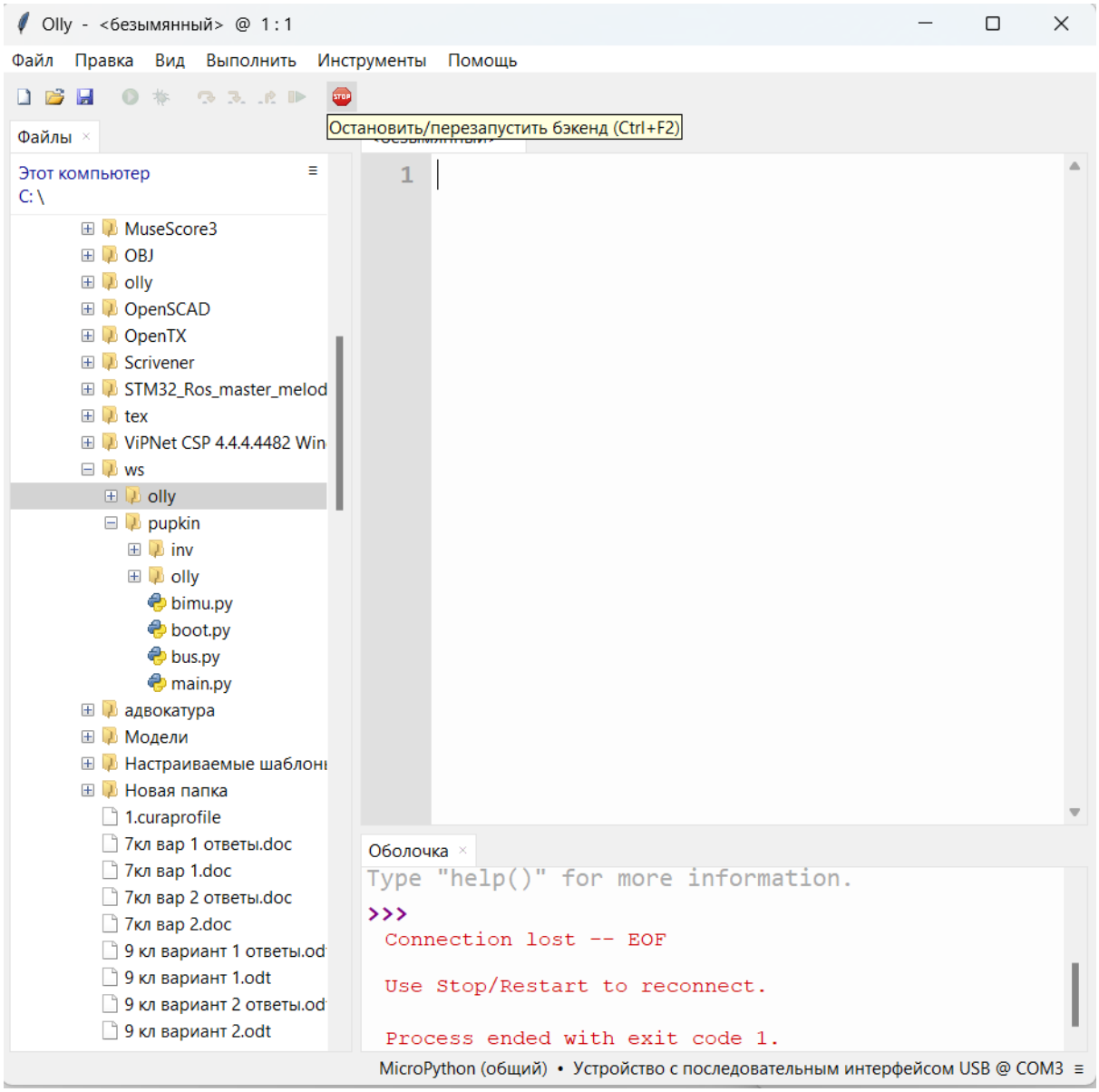


Рис. 1.25: Синхронизация работы IDE Olly и робота

После осуществления синхронизации в самом низу окна панели “Оболочка” появится приглашение к работе среды MicroPython робота в виде трех угловых скобок, а в нижней части панели “Файлы” будет видно содержимое внутренней flash памяти робота под заголовком “Устройство MicroPython” (рисунок 1.26). После этого можно непосредственно переходить к программированию робота.

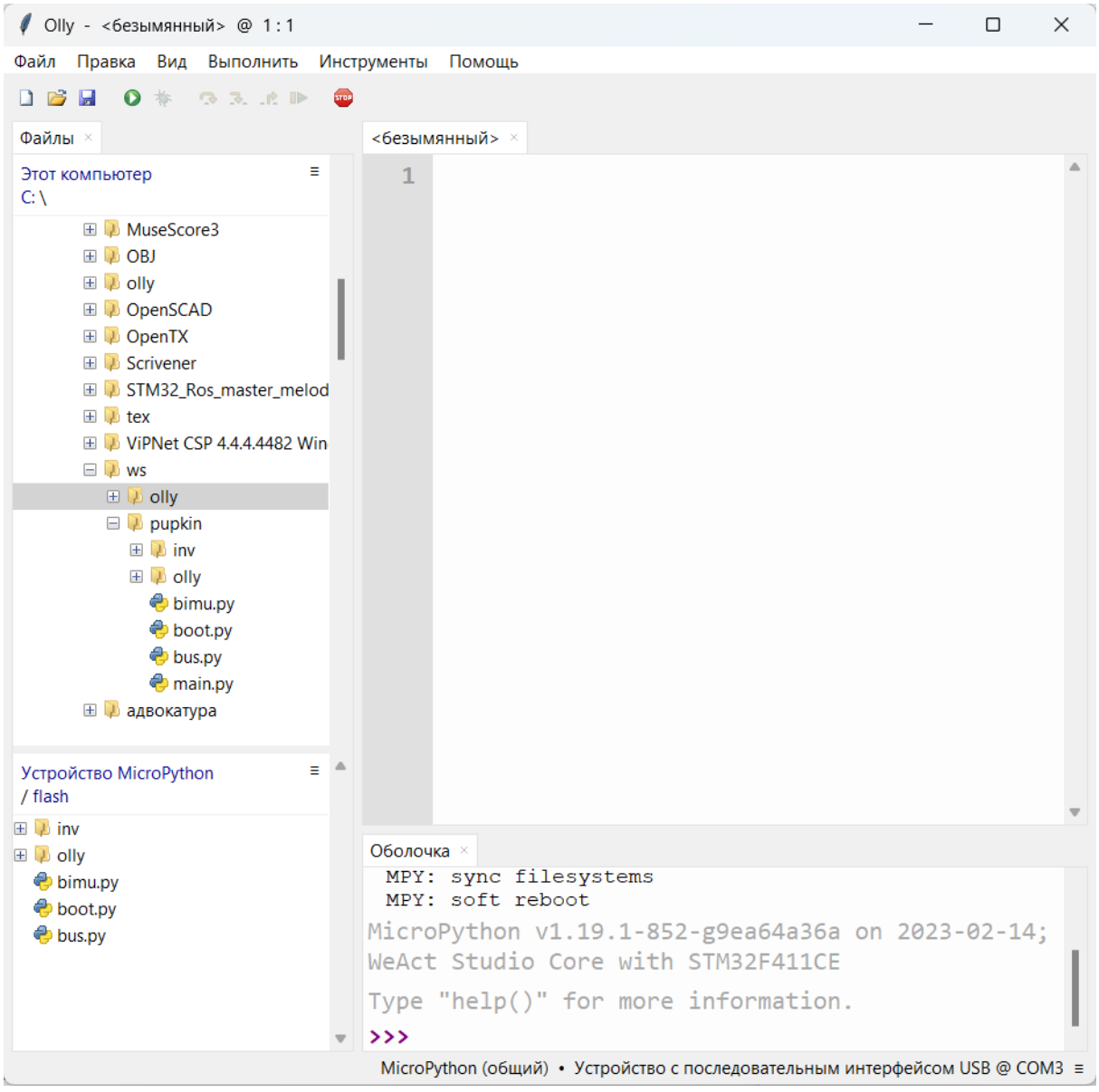


Рис. 1.26: IDE Olly с подключенным роботом

Программирование робота

Для работы программ этого пособия необходимо, чтобы файловая система робота содержала некоторые файлы и каталоги. В частности необходимо, чтобы каталог /flash внутренней памяти робота содержал каталоги inv и olly, а также файлы main.py и boot.py (рисунок 1.27). Если этих файлов и каталогов нет, то следует прошить файловую систему робота согласно приложению Ж.

Если указанные каталоги и файлы присутствуют необходимо найти файл main.py. Именно в этом файле находится код программы управления роботом.

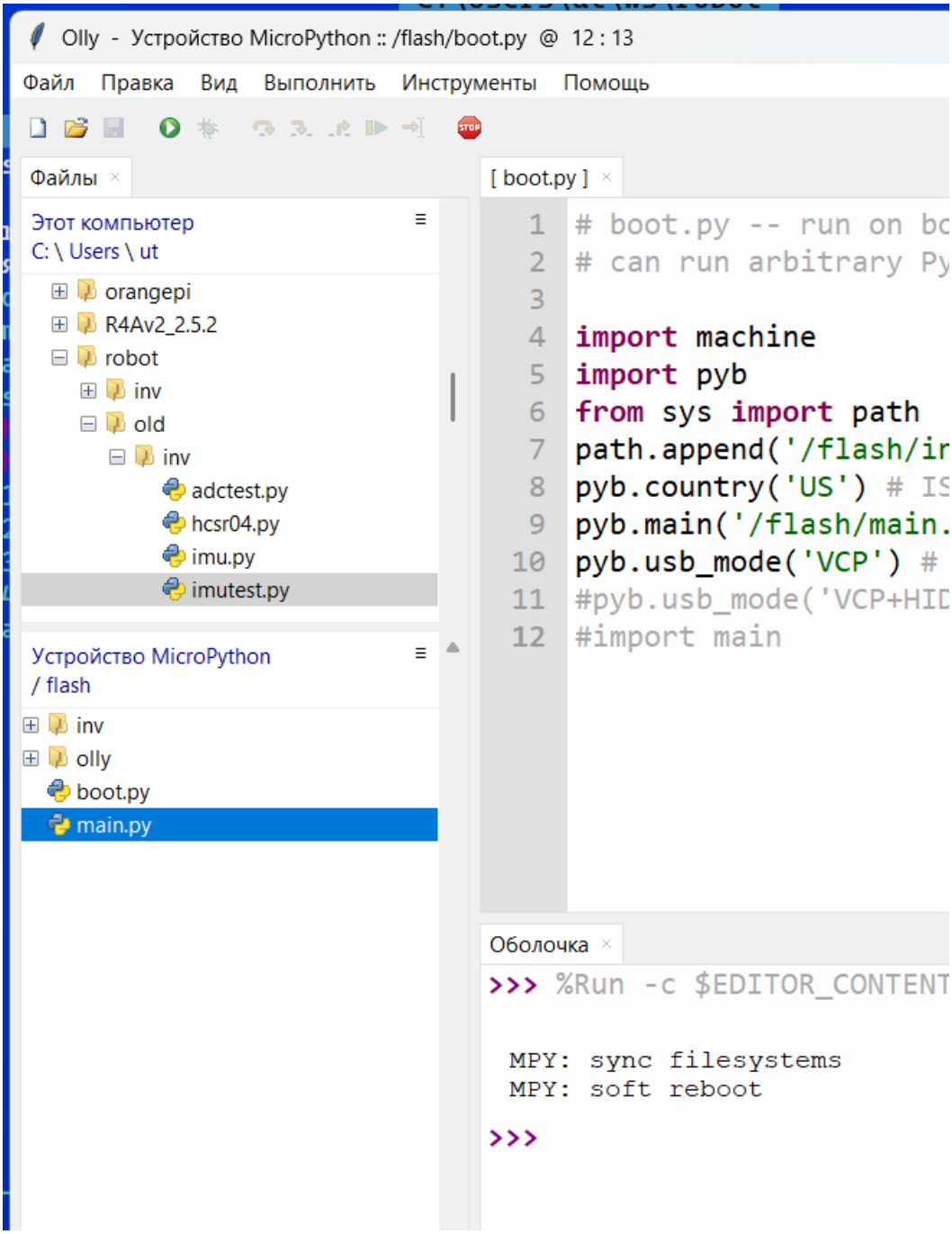


Рис. 1.27: Содержимое внутренней памяти робота

Двойной щелчек мыши по имени этого файла приводит к его открытию в редакторе (рисунок 1.28). После чего код файла может быть изменен.

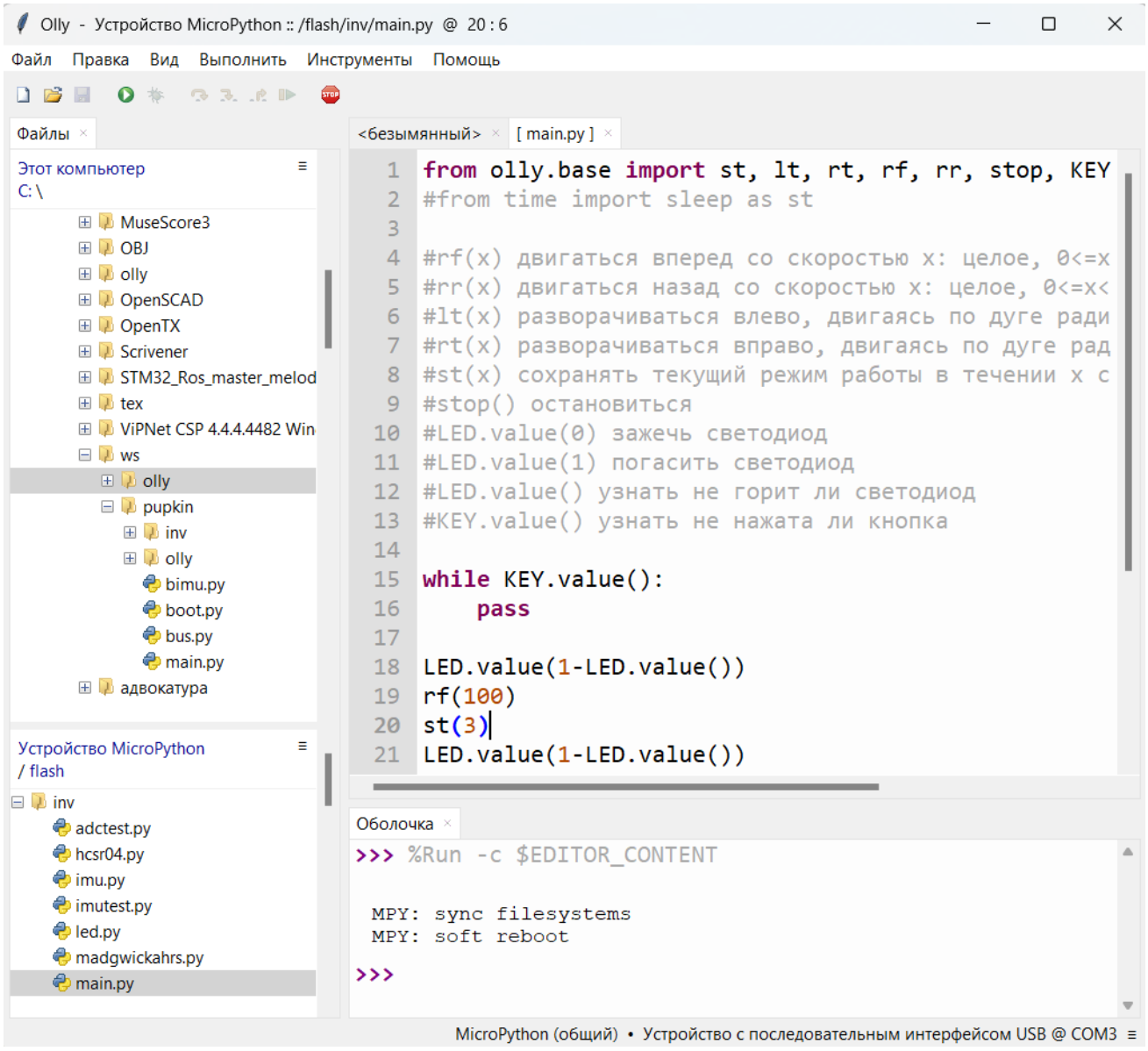


Рис. 1.28: Код программы управления роботом

После того как код будет приведен к желаемому виду необходимо сохранить файл main.py на роботе, для этого надо нажать мышкой на ставшую ярко-синей иконку дискеты в строке кнопок быстрого доступа к командам (рисунок 1.29).

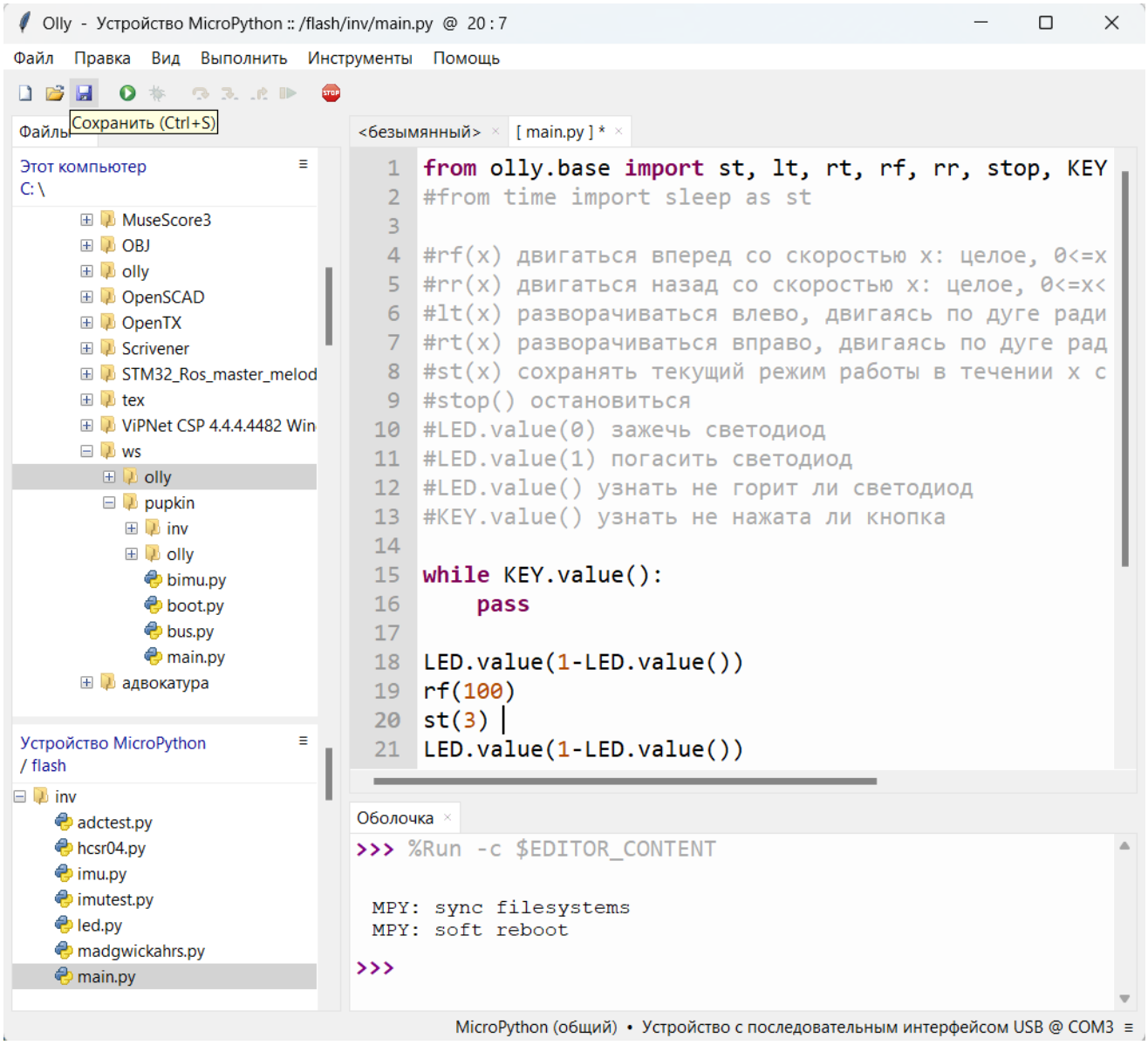


Рис. 1.29: Отредактированный файл с несохраненным кодом программы управления роботом

Нажатие на эту иконку приведет к сохранению файла (рисунок 1.30).

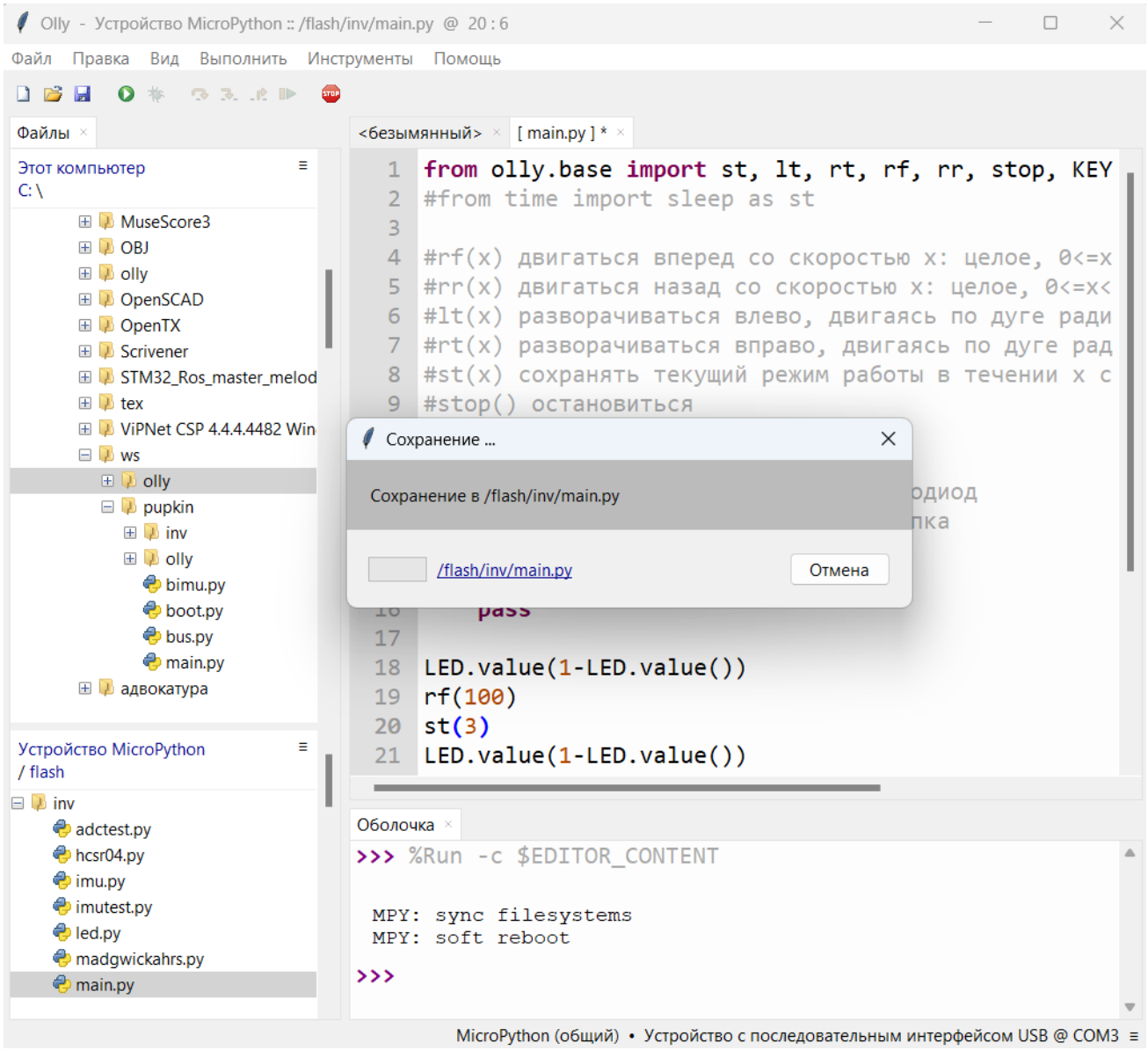


Рис. 1.30: Сохранение main.py

После завершения сохранения файла дискетка станет серой (рисунок 1.31) и робота можно считать запрограммированным, теперь его можно отключить от компьютера, подключить к аккумулятору и запустить программу нажав на нем на кнопку.

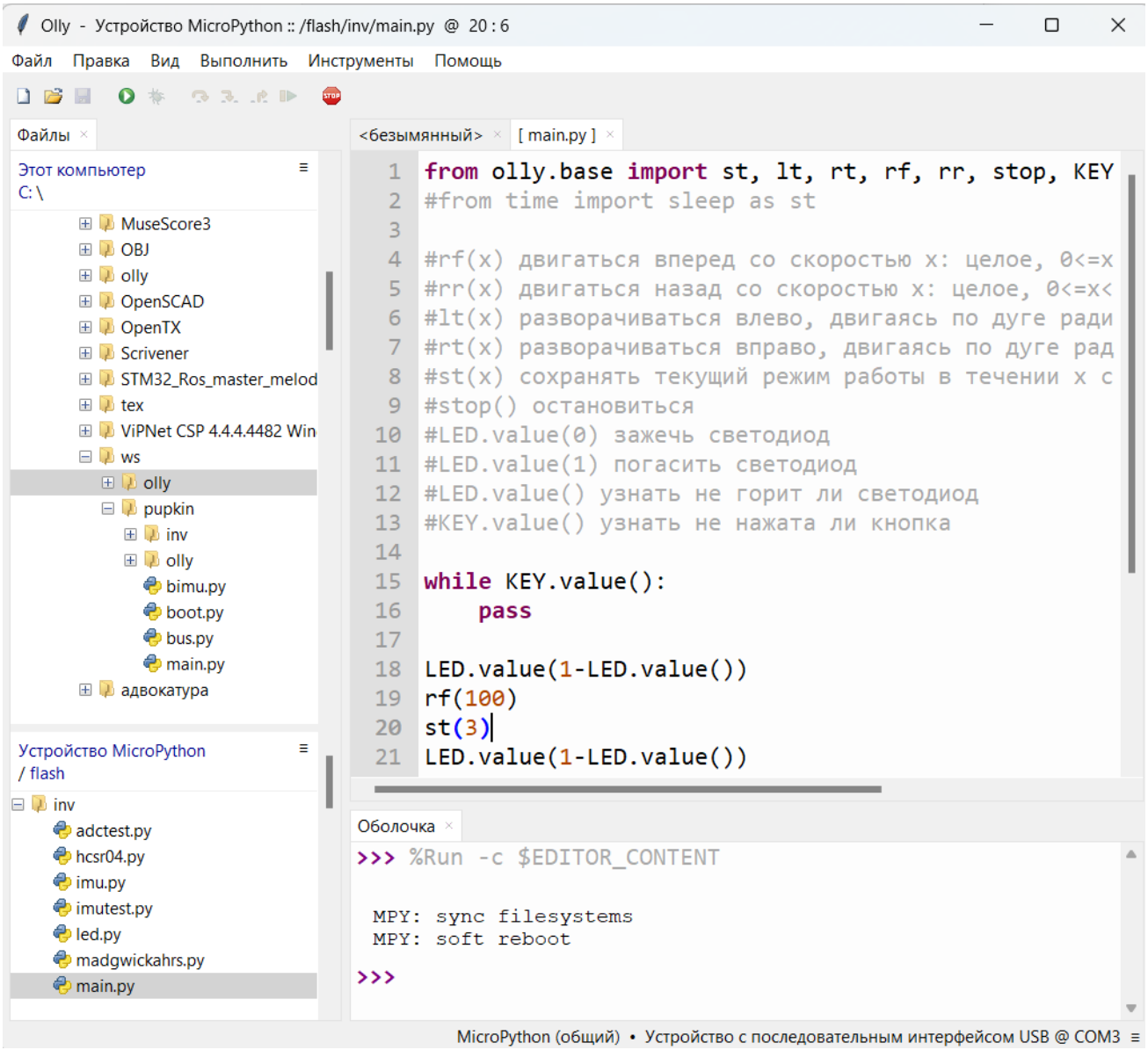


Рис. 1.31: Сохраненный файл main.py

Значимость умения программировать роботов определяется их возрастающей ролью в нашей жизни. Постоянно расширяется их область применения как в мирных целях так и в военной сфере. Это и роботы пылесосы, медицинские роботы, беспилотные такси, это и военные роботы, например, барражирующие воздушные боеприпасы. Положительными сторонами роботов является то, что они никогда не устают, не ленятся и не обманывают.

1.3 Разбор кода 1.2. Команды управления роботом

Роботы пылесосы, умные телевизоры, телефоны, кухонные комбайны, предметы интернета вещей, развивающие игрушки (всевозможные аналоги Lego роботов), спортивные тренажеры с каждым годом все плотнее окружают человека. С одной стороны они делают нашу жизнь жизнь комфортнее и проще. С другой стороны они предъявляют человеку определенные требования, поскольку они существенно сложнее предметов, которыми люди пользовались всего несколько десятилетий назад. Одним из таких требований является наличие навыка практического программирования работы этих устройств. И хотя все перечисленные выше предметы достаточно сильно отличаются друг от друга базовые подходы к их программированию имеют много общего.

Научившись программировать что-то одно гораздо проще освоить программирование чего-то другого. Все дальнейшее содержание книги будет посвящено отработке навыков программирования на примере программирования школьного робота. И начнется эта отработка с внимательного рассмотрения кода 1.2 и команд управления.

В строке 1 кода происходит импортирование из библиотечного модуля `olly.base` необходимых для работы программы элементов (код модуля приведен в приложении Е).

Строки с 3 по 12 являются комментариями - некоторыми пояснениями о работе программы и отдельных ее частей - они нужны для человека, но не обрабатываются MicroPython. Все, что идет в строке после знака #, является комментарием.

В комментариях кода 1.2 перечислены команды, которые понимает описанный в приложениях Б и В робот. Так по команде `rf(x)` (где `x` - целое число от 0 до 100) робот начинает двигаться вперед со скоростью `x` (ориентировочно в процентах от максимальной скорости). Например, по команде `rf(100)` робот начинает двигаться вперед с максимальной скоростью, по команде `rf(90)` - со скоростью 0.9 от максимальной. Следует отметить, что из-за разности характеристик двигателей робот может осуществлять движение не строго по прямой, а с некоторым отклонением, что с легкостью можно компенсировать соответствующими поправками в программе.

Как легко видно из комментариев в строчках 3-12, робот умеет двигаться вперед, назад, вращаться вправо и влево. Кроме того он умеет сохранять текущий режим работы, зажигать и гасить встроенный светодиод, определять состояние кнопки и светодиода. Список команд можно найти в приложении А

Двигаясь вперед или назад (команды `rf` и `rr`) робот будет проходить расстояние которое зависит от времени (задается действительным числом в строке 18) и скорости движения.

По командам `lt(0)` и `rt(0)` робот вращается на месте - радиус разворота равен нулю и робот осуществляет, так называемый, танковый разворот. По командам `lt(1)` и `rt(1)` робот вращается вокруг одного из колес. По командам `lt(2)` и `rt(2)` робот движется по дуге окружности большего радиуса чем в предыдущем случае. По командам `lt(100)` и `rt(100)` робот движется почти по прямой.

При вращениях (команды `lt` и `rt`) угол поворота зависит от радиуса дуги по которой движется робот и времени вращения (задается действительным числом в строке 20).

После выполнения строки 1 кода 1.2 управление переходит к строкам 14-15 в которых программа ждет нажатия кнопки на роботе.

Как только кнопка будет нажата робот начнет движение вперед со скоростью 0.9 от максимально возможной, выполняя команду в строке 17 и будет так двигаться полторы секунды - команда в строке 18.

Затем робот начнет выполнять танковый разворот влево в течении двух секунд (строки 19 и 20), после чего остановится - строка 21.

К слову сказать, в русскоязычной среде наряду со словом *команда* используются слова *оператор* (под которым понимается отдельная команда или набор команд) и *инструкция* (которое лингвистически ближе всего к оригинальному английскому понятию *statement*). Когда говорят о языке программирования или программном коде то чаще употребляют слово *оператор*. В дальнейшем тексте наряду со словом *команда* будет использоваться слова *оператор* и *инструкция*.

Часто при написании программ приходится поправлять их работу — производить отладку. Если работа производится с использованием симулятора робота, то для целей отладки в IDE Olly существует ряд отладочных инструментов, для использования которых необходимо запустить отладчик, нажав на кнопочку с изображением жука на панели инструментов:

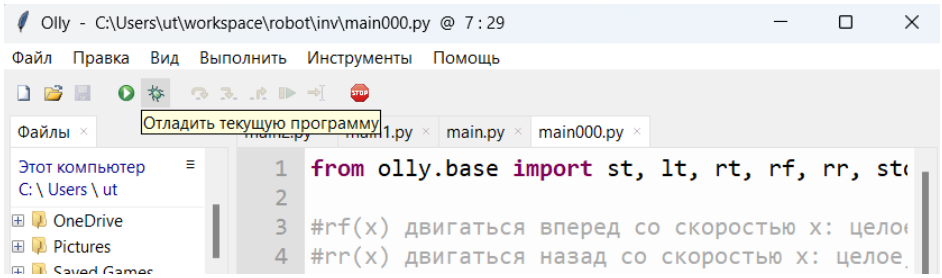


Рис. 1.32: Использование отладчика

После запуска отладчика справа от кнопочки с изображением жука станут активны кнопки с различными инструментами отладки, кроме того в программном коде будет выделен оператор, стоящий первым в очереди на выполнение.

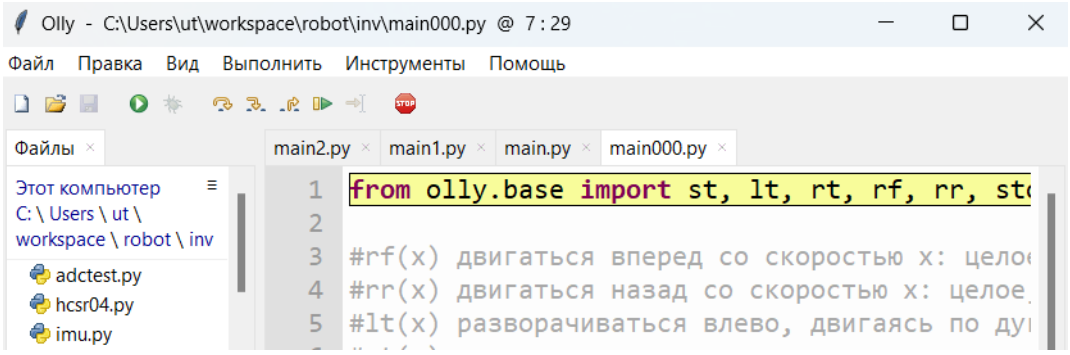


Рис. 1.33: Использование отладчика

Для пооператорного выполнения программы используется инструмент отладчика “Перепрыгнуть”, который можно вызвать либо нажав соответствующую кнопку на панели инструментов, либо нажатием клавиши “F6” на клавиатуре:

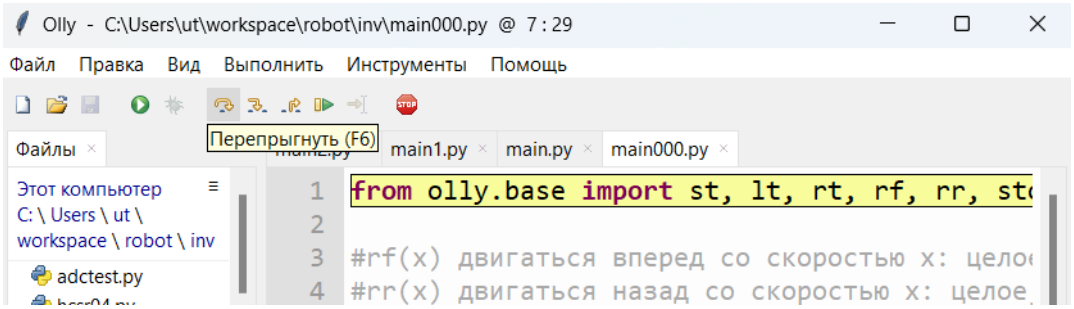


Рис. 1.34: Использование отладчика

При выполнении первого оператора в программе, как было уже указано выше, происходит импортирование необходимых программных объектов из библиотеки `olly.base`, также происходит инициализация симулятора робота и открывается графическое окно симулятора с условным изображением робота:

Рис. 1.35: Использование отладчика

Кроме того, после окончания выполнения оператора первым в очереди на выполнение станет следующий оператор в программе, в рассматриваемом коде — ожидание нажатия кнопки:

```
11 #LED.value() узнать не горит ли светодиод
12 #KEY.value() узнать не нажата ли кнопка
13
14 while KEY.value():
15     pass
16
17 rf(90)
```

Рис. 1.36: Использование отладчика

Для того чтобы выполнить этот оператор необходимо снова нажать кнопку “Перепрыгнуть”:

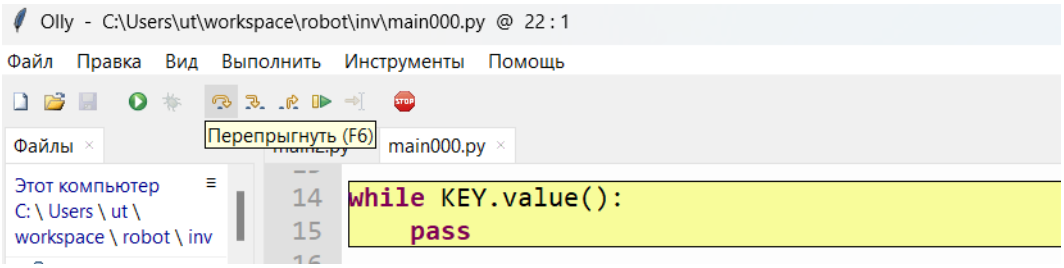


Рис. 1.37: Использование отладчика

После этого кнопки инструментов отладки станут неактивны, а подсветка оператора, стоящего первым в очереди выполнения, не изменится:

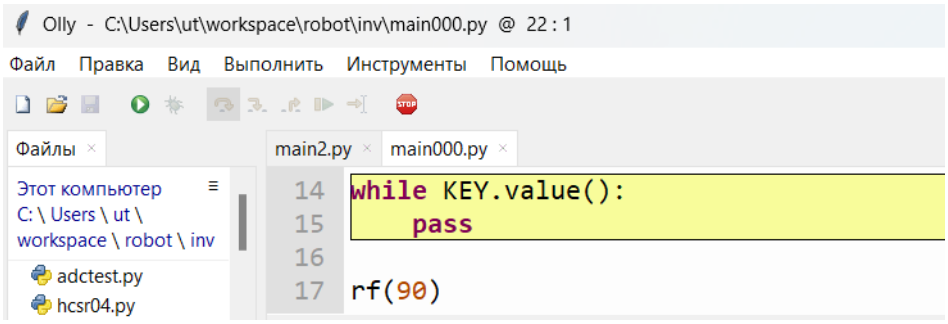


Рис. 1.38: Использование отладчика

Это происходит потому, что этот оператор ожидает нажатия кнопки (в случае симулятора — клавиши “Shift”). После нажатия кнопки выполнение оператора закончится и первым в очереди выполнения станет следующий оператор:

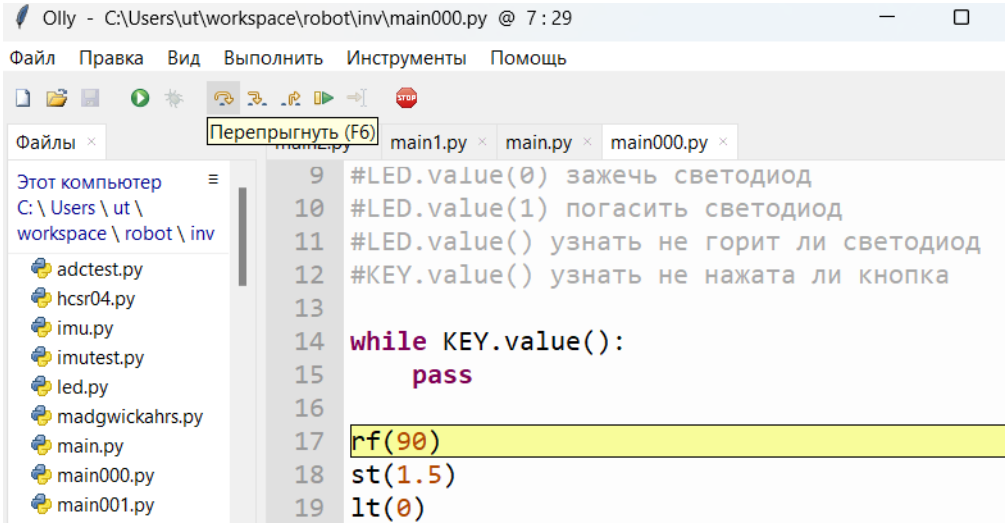


Рис. 1.39: Использование отладчика

После его выполнения с помощью инструмента отладки “Перепрыгнуть” (а это оператор включения двигателей для движения вперед) положение робота не изменится:

Рис. 1.40: Использование отладчика

А активным станет следующий оператор:

```
16
17 rf(90)
18 st(1.5)
19 lt(0)
20 st(2)
```

Рис. 1.41: Использование отладчика

И только при выполнении этого оператора (формирования задержки между командами управления роботом) робот в графическом окне симулятора осуществит движение вперед:

Рис. 1.42: Использование отладчика

А управление в программе перейдет к следующей команде:

```
17 rf(90)
18 st(1.5)
19 lt(0)
20 st(2)
21 stop()
```

Рис. 1.43: Использование отладчика

Это команда включения двигателей для осуществления танкового разворота налево, после ее выполнении робот опять не изменит своего положения:

Рис. 1.44: Использование отладчика

А первым в очереди выполнения станет оператор формирования задержки, стоящий в 20 строке программы:

```
18 st(1.5)
19 lt(0)
20 st(2)
21 stop()
22
```

Рис. 1.45: Использование отладчика

Именно во время выполнения этого оператора робот осуществит разворот:

Рис. 1.46: Использование отладчика

И в программе останется только один невыполненный оператор — оператор остановки робота:

```
19 lt(0)
20 st(2)
21 stop()
22
```

Рис. 1.47: Использование отладчика

При очередном применении инструмента отладки “Перепрыгнуть” этот оператор начнет выполняться и во время выполнения будет ожидать закрытия графического окна симулятора, при этом инструменты отладки станут неактивны:

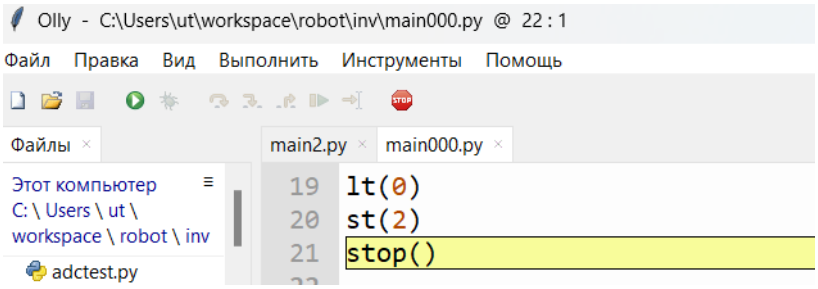


Рис. 1.48: Использование отладчика

Для завершения выполнения этого оператора необходимо закрыть графическое окно симулятора, нажав мышкой на крестик в правом верхнем углу этого окна:

Рис. 1.49: Использование отладчика

После закрытия графического окна симулятора выполнение программы закончится:

```
19 lt(0)
20 st(2)
21 stop()
22
```

Рис. 1.50: Использование отладчика

Пошаговое (пооператорное) выполнение программы является отличным инструментом для выполнения анализа кода и нахождения логических ошибок в алгоритме. Логические ошибки — ошибки в программе, которые не являются синтаксическими ошибками. Для поиска синтаксических ошибок в программе можно воспользоваться панелью “Оболочка” IDE Olly:

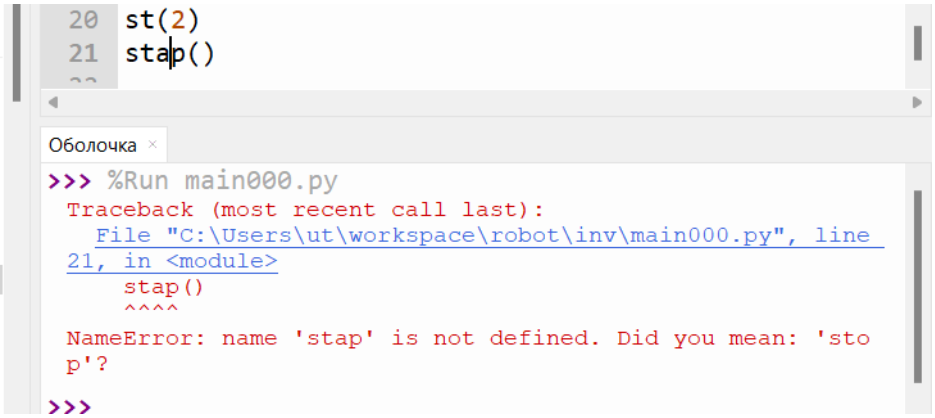


Рис. 1.51: Диагностическое сообщение в панели “Оболочка”

В этом примере в строке 21 допущена синтаксическая ошибка. При попытке выполнения этой программы ее работа прерывается и в панели “Оболочка” выводится диагностическое сообщение с указанием примерного места расположения ошибки.

Для того чтобы панель “Оболочка” была видна в IDE Olly необходимо, чтобы напротив пункта “Оболочка” в всплывающем меню “Вид” стояла галочка:

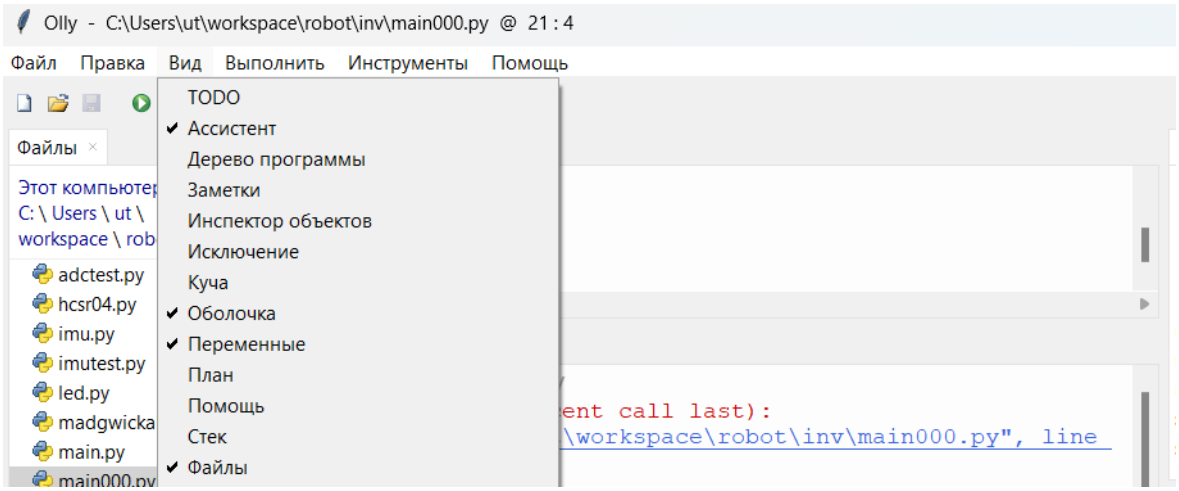


Рис. 1.52: Активация панели “Оболочка”

Зная команды управления роботом и то как они работают легко синтезировать алгоритм, заставляющий робота выполнить желаемую задачу. Такой синтез должен начинаться с анализа задачи. Пусть необходимо, чтобы робот проехал по следующей траектории:

Рис. 1.53: Заданная траектория

Видно, что вначале робот едет прямо, а в конце прямолинейного отрезка движения поворачивает налево на 90°и повторяет такую последовательность движений 4 раза. Соответственно для начала необходимо, чтобы робот научился ехать прямо и поворачивать налево на 90°. Для этого в коде:

Код 1.3: Синтез алгоритма движения по квадрату

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  rf(90)
7  st(1.5)
8  lt(0)
9  st(2)
10 stop()
```

необходимо подобрать значение времени задержки в 9 строке. При значении времени задержки около 1 секунды:

Код 1.4: Синтез алгоритма движения по квадрату

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  rf(90)
7  st(1.5)
8  lt(0)
9  st(1)
10 stop()
```

угол поворота составит примерно 90°:

Рис. 1.54: Синтез алгоритма движения по квадрату

Эта траектория движения робота составляет четверть от необходимого квадрата. Для того чтобы получить полный квадрат необходимо строки 6 — 9 повторить 4 раза:

Код 1.5: Синтез алгоритма движения по квадрату

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3
4
5  while KEY.value():
6      pass
7
8  rf(90)
9  st(1.5)
10 lt(0)
11 st(1)
12 rf(90)
13 st(1.5)
14 lt(0)
15 st(1)
16 rf(90)
17 st(1.5)
18 lt(0)
19 st(1)
20 rf(90)
21 st(1.5)
22 lt(0)
23 st(1)
24 stop()
```

Двигаясь под управлением этой программы робот проедет по следующей траектории:

Рис. 1.55: Синтез алгоритма движения по квадрату

Видно, что идеального квадрата не получилось. Это связано с тем, что при времени разворота в 1 секунду робот успевает развернуться на угол чуть меньший 90°. Подобрав более точно значение времени разворота в строках 11, 15, 19 и 23 можно получить идеальный квадрат:

Рис. 1.56: Синтез алгоритма движения по квадрату

Задания

1. Наберите код 1.2 и выполните его либо в симуляторе Olly, либо на роботе “Ветеран”. Опишите поведение робота. Обратите внимание, что для повторного запуска робота “Ветеран” необходимо отключить аккумулятор и заново его подключить. Выполните программу пошагово в симуляторе. При выполнении какого оператора робот передвигается вперед? При выполнении какого оператора робот осуществляет разворот?

2. Изучите действие команд `rf`, `rr`, поменяв в модуле main 17 строку на:

   - `rf(0)`
   - `rf(5)`
   - `rf(25)`
   - `rf(50)`
   - `rf(75)`
   - `rf(100)`
   - `rr(0)`
   - `rr(5)`
   - `rr(25)`
   - `rr(50)`
   - `rr(75)`
   - `rr(100)`

   Сравните поведение робота при различных командах в 17 строке программы. Выполните обобщение по проведенным наблюдениям — что и как влияет на поведение робота.

3. Верните модуль main в исходное состояние. Изучите действие команды `st`, поменяв 18 строку на:

   - `st(0)`
   - `st(0.5)`
   - `st(1)`
   - `st(1.5)`
   - `st(2)`
   - `st(2.5)`
   - `st(3)`

   Сравните поведение робота при различных командах в 18 строке программы. Выполните обобщение по проведенным наблюдениям — что и как влияет на поведение робота.

4. Изучите действие команд `lt`, `rt`, поменяв в модуле main 19 строку на:

   - `lt(0)`
   - `lt(1)`
   - `lt(2)`
   - `lt(3)`
   - `lt(50)`
   - `lt(100)`
   - `rt(0)`
   - `rt(1)`
   - `rt(2)`
   - `rt(3)`
   - `rt(50)`
   - `rt(100)`

   Сравните поведение робота при различных командах в 19 строке программы. Выполните обобщение по проведенным наблюдениям — что и как влияет на поведение робота.

5. Верните модуль main в исходное состояние. Изучите действие команды `st`, поменяв 20 строку на:

   - `st(0)`
   - `st(0.5)`
   - `st(1)`
   - `st(1.5)`
   - `st(2)`
   - `st(2.5)`
   - `st(3)`

Сравните поведение робота при различных командах в 20 строке программы. Выполните обобщение по проведенным наблюдениям — что и как влияет на поведение робота.

6. Верните модуль main в исходное состояние (код 1.2). Удалите из него 18 строчку. Объясните изменение поведения робота.

7. Верните модуль main в исходное состояние. Удалите из него 20 строчку. Объясните изменение поведения робота.

8. Верните модуль main в исходное состояние. Удалите из него 21 строчку. Объясните изменение поведения робота. Следует отметить, что поведение настоящего робота и робота в симуляторе будут в этом случае отличаться.

9. Измените код 1.2 таким образом, чтобы робот делал не левый а правый поворот.

10. Измените итоговый код предыдущей задачи так, чтобы робот делал правый поворот на 90°.

11. Измените итоговый код предыдущей задачи так, чтобы робот проехал по квадрату. Квадрат будет считаться выполненым, если все углы в нем будут одинаковыми, а точка старта робота будет совпадать с точкой финиша. При редактировании кода используйте комбинации клавиш “Ctrl”+“X”, “Ctrl”+“C” и “Ctrl”+“V”.

1.4 Переменные, оператор присваивания

В параграфе 1.3 была синтезирована следующая программа для движения робота по квадрату:

Код 1.6: Вариант кода модуля main для движения по квадрату

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3
4
5  while KEY.value():
6      pass
7
8  rf(90)
9  st(1.5)
10 lt(0)
11 st(1)
12 rf(90)
13 st(1.5)
14 lt(0)
15 st(1)
16 rf(90)
17 st(1.5)
18 lt(0)
19 st(1)
20 rf(90)
21 st(1.5)
22 lt(0)
```

```
23 st(1)
24 stop()
```

Для настройки программы надо настроить целых четыре команды - в строках 11, 15, 19 и 23. А именно: надо настроить длительность четырех поворотов. Можно написать код, который будет требовать настройки только одной команды:

Код 1.7: Упрощенный вариант кода модуля main для движения по квадрату

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2  #from time import sleep as st
3
4
5  while KEY.value():
6      pass
7  g=1
8  rf(90)
9  st(1.5)
10 lt(0)
11 st(g)
12 rf(90)
13 st(1.5)
14 lt(0)
15 st(g)
16 rf(90)
17 st(1.5)
18 lt(0)
19 st(g)
20 rf(90)
21 st(1.5)
22 lt(0)
23 st(g)
24 stop()
```

В этом варианте кода используется специальный программный объект, называемый переменная. Переменные умеют хранить данные, т.е. говорят, что они имеют значение. Переменные имеют имя, по этому имени в программе можно получить доступ к хранимым в переменных данным - к значениям переменных. Значения переменных при необходимости можно менять. Смена значения переменных производится с помощью команды присваивания. Выражения *команда присваивания* и *оператор присваивания* обозначают практически одно и то же.

В коде 1.7 присутствует переменная `g`. В строке 7 этой переменной с помощью оператора присваивания `=` присваивается значение, которое в дальнейшем используется в командах в строках 11, 15, 19 и 23.

Проиллюстрировать сказанное можно с помощью отладчика. Для начала необходимо его запустить нажав на кнопку с изображение жука:

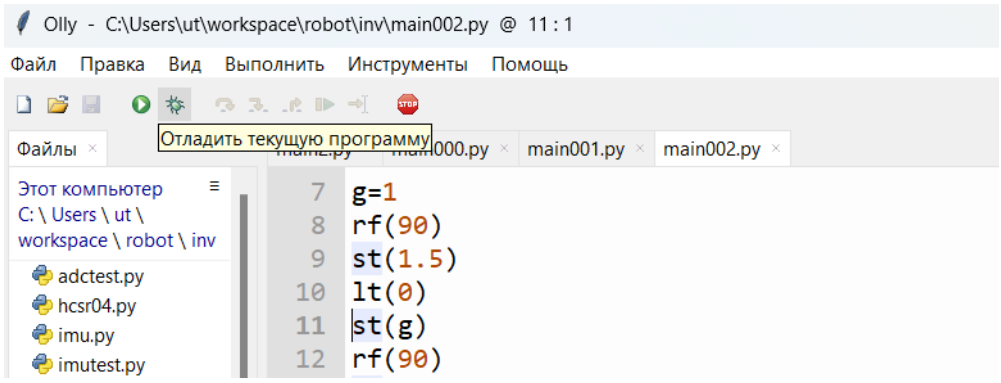


Рис. 1.57: Использование переменных

Далее для пошагового выполнения можно воспользоваться отладочным инструментом “Перепрыгнуть”:

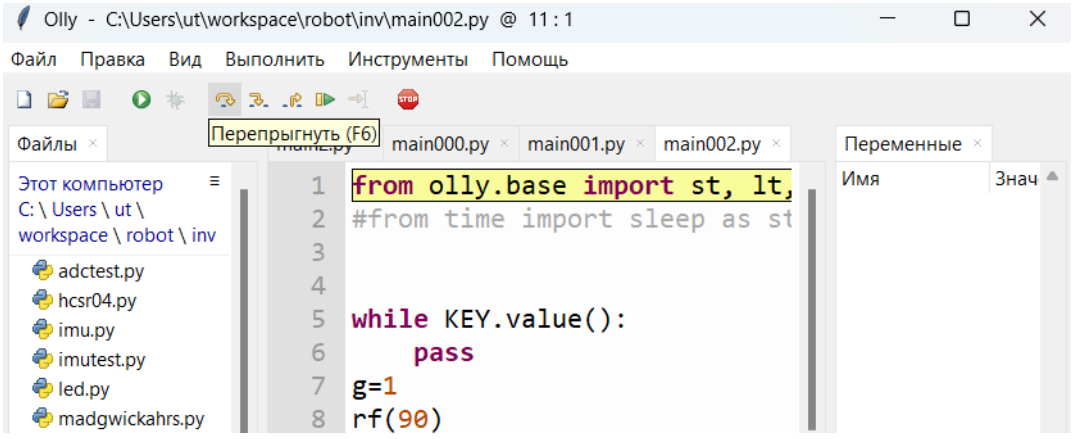


Рис. 1.58: Использование переменных

После импортирования необходимых объектов и открытия графического окна симулятора школьного робота первым в очереди на выполнене станет оператор ожидания нажатия кнопки, запуск выполнения которого также можно осуществить с помощью инструмента “Перепрыгнуть”:

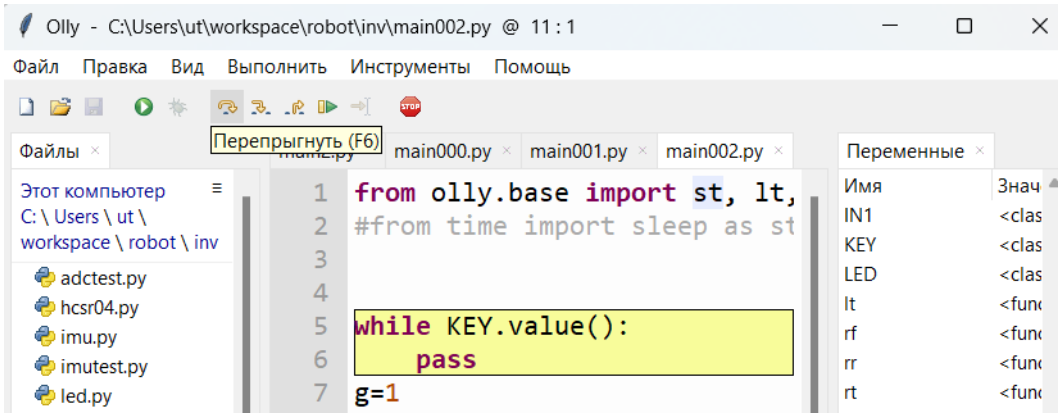


Рис. 1.59: Использование переменных

Во время выполнения этого оператора кнопки инструментов отладки станут неактивными, а сам оператор будет выполняться до тех пор пока не будет нажата клавиша “Shift” на клавиатуре:

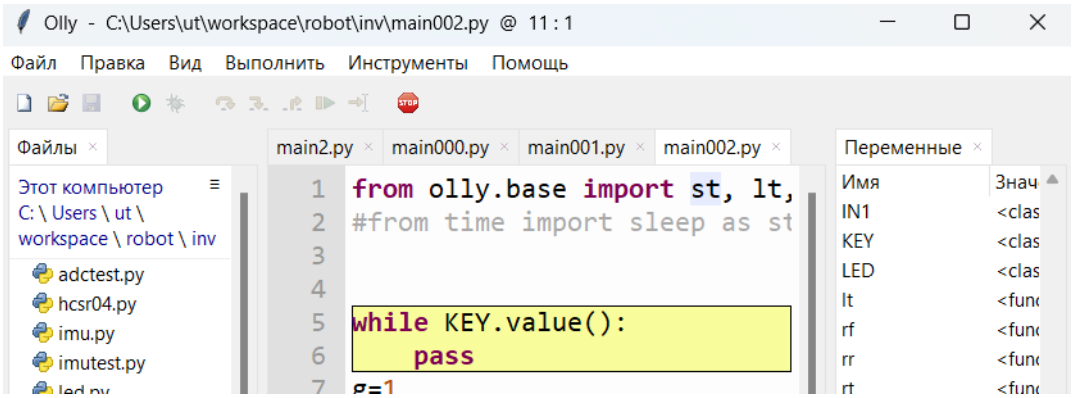


Рис. 1.60: Использование переменных

После нажатия клавиши “Shift” выполнение этого оператора прекратится, а первым в очереди на выполнение станет следующий оператор — оператор присваивания значения переменной g.

До выполнения этого оператора, как легко убедиться в панели ”Переменные”, значение этой переменной не определено — в правой панели ее нет в списке:

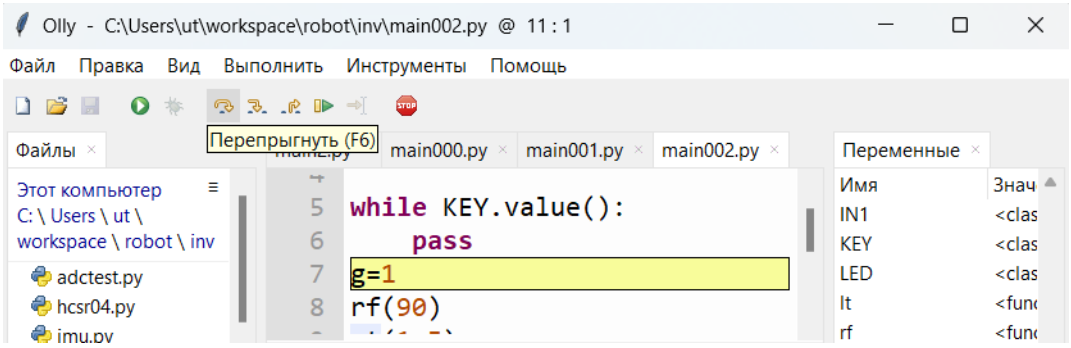


Рис. 1.61: Использование переменных

Для того чтобы панель “Переменные” была видна в IDE Olly необходимо, чтобы напротив пункта “Переменные” в всплывающем меню “Вид” стояла галочка:

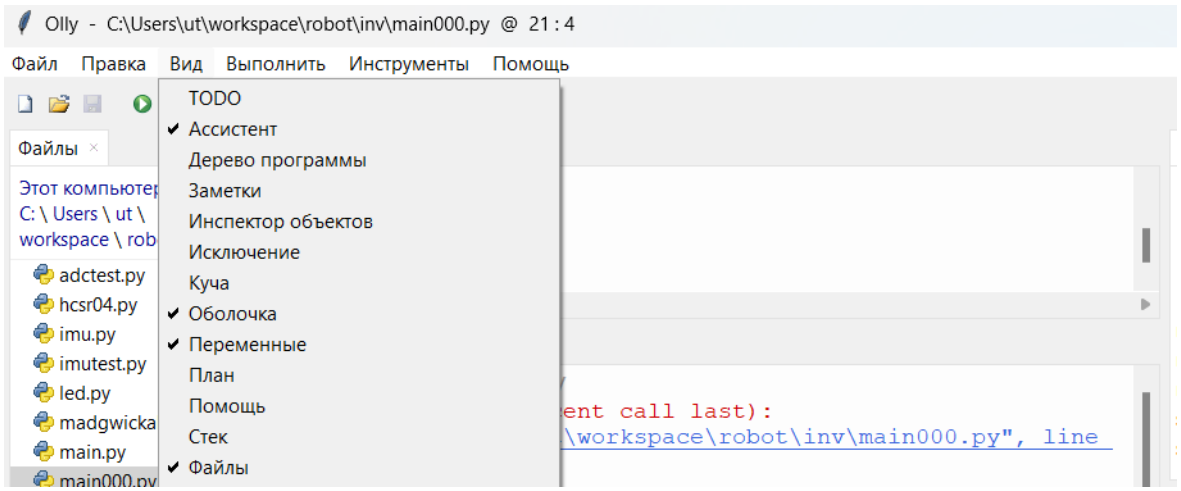


Рис. 1.62: Использование переменных

После выполнения оператора присваивания значение переменной g станет равно 1 и она появится в списке панели “Переменные” вместе со своим значением:

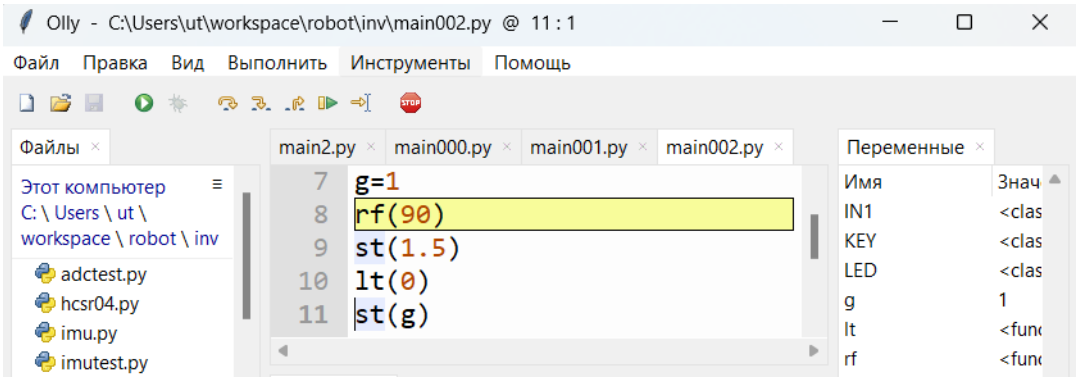


Рис. 1.63: Использование переменных

Далее это значение можно использовать в 11, 15, 19 и 23 строках программы. Для того чтобы быстрее дойти до выполнения этих строк можно воспользоваться инструментом отладка “Выполнить до курсора”. Для этого курсор необходимо поставить в нужную строку программы и нажать соответствующую кнопку панели инструментов:

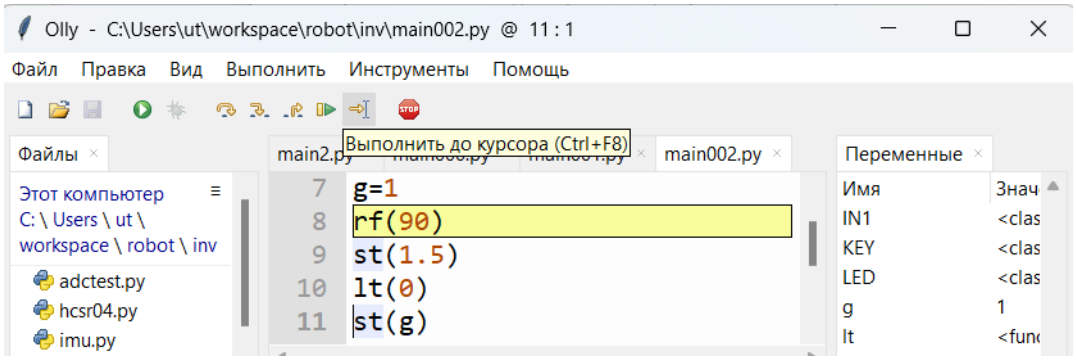


Рис. 1.64: Использование переменных

После использования этого инструмента первым в очереди на выполнение станет оператор перед которым стоит курсор, при этом видно, что значение переменной g равно 1 и именно это значение будет использовано при формировании задержки в 11 строке программы:

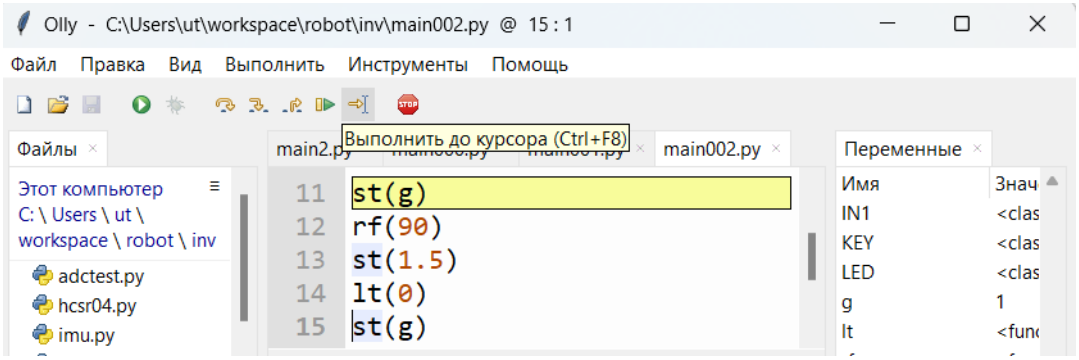


Рис. 1.65: Использование переменных

После помещения курсора в начало 15 строки и нового вызова инструмента “Выполнить до курсора” первым в очереди на выполнение станет оператор в 15 строке, а значение переменной g останется равным 1:

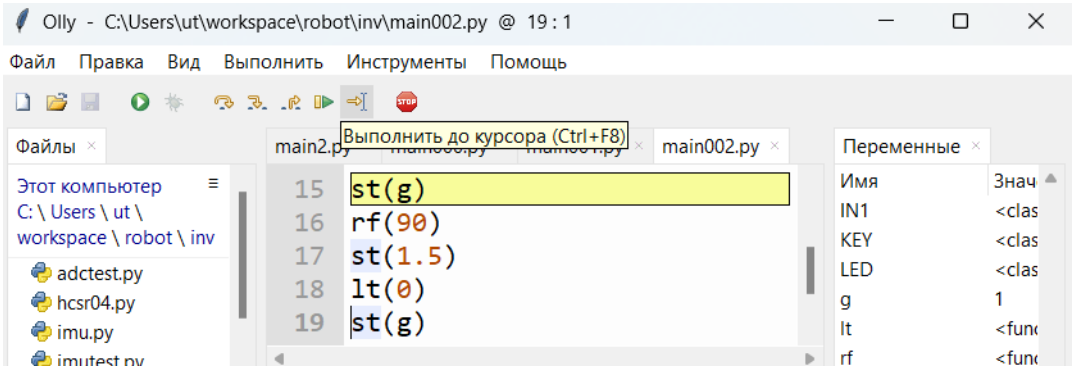


Рис. 1.66: Использование переменных

Выполнив эту процедуру еще два раза можно убедиться, что в 19 и в 23 строке будет использовано то же самое значение переменной g.

Таким образом, в рассматриваемом варианте кода модуля main требуется настроить только одну команду — экспериментальным путем подобрать значение переменной g в строке 7 такое, чтобы робот двигался по квадрату. При значении этой переменной равном 1 робот проедет по следующей траектории:

Рис. 1.67: Использование переменных

Видно, что это значение маловато — робот немного не доворачивает, если увеличить значение переменной до 1.1 получится следующая траектория:

Рис. 1.68: Использование переменных

Это значение слишком велико. Идеальный квадрат можно получить, если значение переменной сделать равным 1.05:

Рис. 1.69: Использование переменных

Задания

1. Ознакомьтесь в каком-либо учебнике по Python с правилами присвоения имен.
2. Ознакомьтесь в каком-либо учебнике по Python с типами переменных.
3. Запрограммируйте робота, используя код 1.7. Экспериментально подберите значение переменной `g` так, чтобы робот ехал по квадрату.
4. Перепишите код 1.7 так, чтобы робот делал не левый, а правый поворот. Объясните почему значение переменной `g` могло измениться.
5. Перепишите код 1.7 так, чтобы изменяя только одну команду программы, можно было изменять длину стороны квадрата.
6. Запрограммируйте робота для движения по окружности (рисунок 1.70).

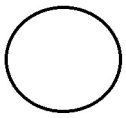


Рис. 1.70: Окружность

7. Запрограммируйте робота для движения по треугольной змейке (рисунок 1.71).

Рис. 1.71: Треугольная змейка

8. Запрограммируйте робота для движения по квадратной змейке (рисунок 1.72).

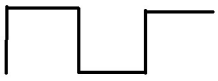


Рис. 1.72: Квадратная змейка

9. Запрограммируйте робота для движения по восьмерке (рисунок 1.73).

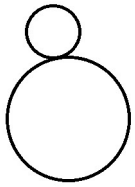


Рис. 1.73: Восьмерка

10. Запрограммируйте робота для движения по квадратной восьмерке (рисунок 1.74).

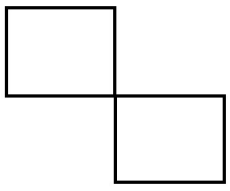


Рис. 1.74: Квадратная восьмерка

1.5 Управление светодиодом

Часто бывает необходимо во время работы робота, чтобы он сообщал о своем состоянии. Для используемого робота можно воспользоваться встроенным в плату микроконтроллера светодиодом. В следующем коде робот движется по квадрату и зажигает светодиод во время движения по каждой четной стороне:

Код 1.8: Движение по квадрату с зажиганием светодиодом при движении вдоль четных сторон

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED



while KEY.value():
    pass
g=1.05
rf(100)
st(1.5)
lt(0)
st(g)
LED.value(0)
rf(100)
st(1.5)
lt(0)
st(g)
LED.value(1-LED.value())
rf(100)
st(1.5)
lt(0)
st(g)
LED.value(1-LED.value())
rf(100)
st(1.5)
lt(0)
st(g)
LED.value(1)
stop()
```

Для зажигания светодиода используется команда:

```
LED.value(0)
```

Для того чтобы погасить светодиод можно использовать:

```
LED.value(1)
```

Для того чтобы инвертировать состояние светодиода (погасить горящий и зажечь не горящий) используется:

```
LED.value(1-LED.value())
```

Задания

1. Запрограммируйте робота, используя код 1.8. Добейтесь его движения по квадрату.

2. Модифицируйте код 1.8, чтобы робот двигался по правильному пятиугольнику (рисунок 1.75) и зажигал светодиод при движении вдоль нечетных сторон.

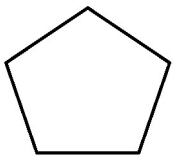


Рис. 1.75: Правильный пятиугольник

Задание по главе

Модифицируйте код 1.8, чтобы робот двигался по звезде (рисунок 1.76) и зажигал светодиод при движении вдоль нечетных сторон.

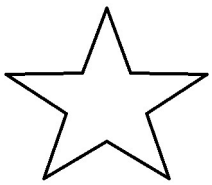


Рис. 1.76: Звезда

Глава 2

Алгоритмические конструкции

2.1 Следование

Ниже представлен некоторый длинный код:

Код 2.1: Длинный код

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
g=0.5
LED.value(0)
rf(100)
st(1.5)
rt(0)
st(g)
LED.value(1)
rf(100)
st(1.5)
lt(0)
st(g)
LED.value(0)
rf(100)
st(1.5)
rt(0)
st(g)
LED.value(1)
rf(100)
st(1.5)
lt(0)
st(g)
LED.value(0)
rf(100)
st(1.5)
rt(0)
st(g)
LED.value(1)
rf(100)
st(1.5)
lt(0)
```

```
36 st(g)
37 stop()
```

В этом коде команды с 6 по 37 следуют друг за другом в простой линейной последовательности. Такая схема соответствует линейным алгоритмам и называется алгоритмической конструкцией следование. Алгоритмическая конструкция следование схематически может быть представлена в виде алгоритмического блока:

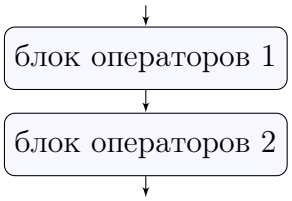


Рис. 2.1: Алгоритмический блок следование

Под блоком операторов понимается как линейная последовательность любых операторов, так и один любой оператор.

Графически код 2.1 может быть представлен укрупненной блок-схемой, представленной на рисунке 2.2.

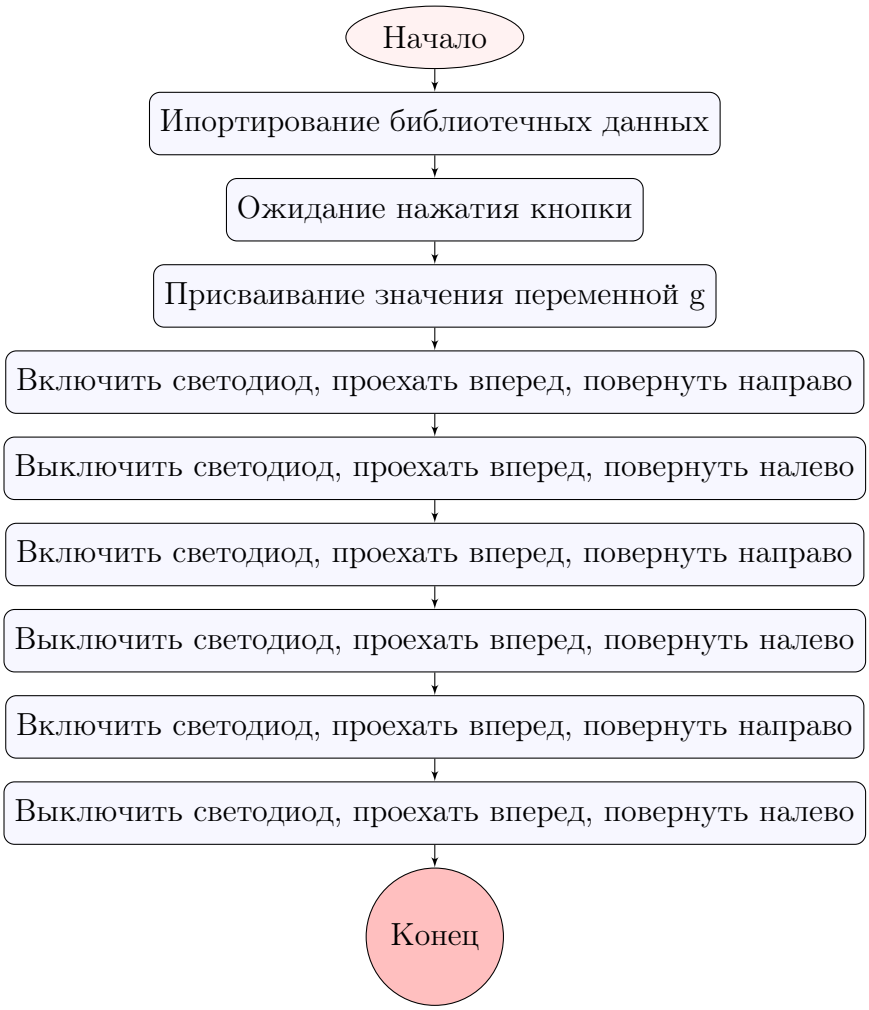


Рис. 2.2: Блок-схема линейного алгоритма

Блок-схемы представляют из себя графический способ записи алгоритмов, они показывают в какой последовательности происходит выполнение команд.

Достоинством линейных алгоритмов является простота, а недостатком - большой размер, а чем больше размер алгоритма тем легче в нем допустить ошибку и тем сложнее его отлаживать.

Задание

1. Ознакомтесь с ГОСТ 19.701-90.

2.2 Повторение, цикл while

Внимательно присмотревшись к коду 2.1, можно обнаружить, что он содержит три одинаковых повторяющихся блока, которые идут друг за другом:

```
LED.value(0)
rf(100)
st(1.5)
rt(0)
st(g)
LED.value(1)
rf(100)
st(1.5)
lt(0)
st(g)
```

Соответственно, код 2.1 можно переписать в более коротком виде, если каким то образом удастся указать, что эти строки нужно повторить несколько раз. Для этого в языке Python имеются определенные синтаксические конструкции - операторы цикла. Например, на основе оператора цикла `while` предыдущий код может быть записан следующим образом:

Код 2.2: Использование цикла `while`

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3
4  while KEY.value():
5      pass
6  g=0.5
7  i=0
8  while i<3:
9      LED.value(0)
10     rf(100)
11     st(1.5)
12     rt(0)
13     st(g)
14     LED.value(1)
15     rf(100)
16     st(1.5)
17     lt(0)
18     st(g)
19     i=i+1
20 stop()
```

Алгоритмы при записи которых используются операторы цикла называются циклическими, а соответствующая алгоритмическая конструкция - повторение.

Оператор цикла `while` схематически может быть представлен в виде алгоритмического блока:

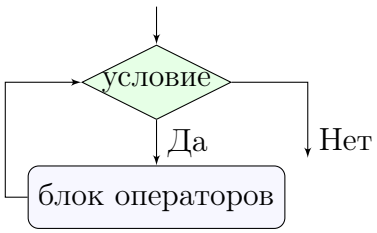


Рис. 2.3: Алгоритмический блок while

Соответственно код 2.2 может быть графически представлен блок-схемой, изображенной на рисунке 2.4.

Рис. 2.4: Блок-схема с циклом while

Структура оператора цикла `while` имеет следующий вид:

```
while условие:
    блок операторов
```

Цикл `while` называется циклом с предусловием. Вначале проверяется `условие`, если оно истинно, то выполняется `блок операторов`, потом опять проверяется условие и так повторяется до тех пор, пока условие не станет ложным, после чего управление переходит к следующей команде после цикла.

В коде 2.2 переменная `i` называется параметром цикла, внутри цикла она пробегает значения от 0 до 2. Эта переменная фактически является счетчиком повторений тела цикла. Необходимо отметить, что после окончания цикла значение этой переменной в данном примере будет равно 3 - количеству повторений тела цикла.

Важно иметь в виду, что все команды составляющие `блок операторов` печатаются с одинаковым отступом от начала строки и образуют тело цикла. Это легко обеспечить применением клавиши “Tab” на клавиатуре.

Ниже представлен еще один вариант алгоритма для движения по квадрату — с применением алгоритмической конструкции повторение:

Код 2.3: Движение по квадрату

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5  i=0
6  while i<4:
7      LED.value(1-LED.value())
8      rf(90)
9      st(1.5)
10     lt(0)
11     st(1.05)
12     i=i+1
13 stop()
```

При движении робот зажигает светодиод на нечетных сторонах квадрата. Видно, что вариант с применением цикла гораздо короче и проще, чем рассмотренные в первой главе варианты.

Выражения

В коде 2.2 есть два объекта на которые следует обратить внимание. Это `i<3` и `i+1`. Конструкции такого вида называются выражениями и служат для обработки информации. Выражениями являются, например, конструкции из переменных и констант, которые соединены между собой знаками: арифметических операций (`+, -, /, *, %, //, **`); сравнения (`>, <, ==, >=, <=, !=`); логических операций (`and, or, not`).

Выражения могут использоваться в командах присваивания справа от знака `=` (как в строке 19 кода 2.2), или в условиях (как в строке 8). При этом в случае команды присваивания вначале производится вычисление выражения, потом осуществляется присваивание. В случае использования выражения в условиях вначале вычисляется выражение, потом происходит проверка условия. Порядок вычисления выражений - слева направо, с учетом приоритета операций, скобки меняют приоритет.

Хорошее правило: если не известен приоритет операций надо использовать скобки.

В качестве примера применения выражений ниже рассмотрена задача синтеза алгоритма движения робота по расходящейся спирали:

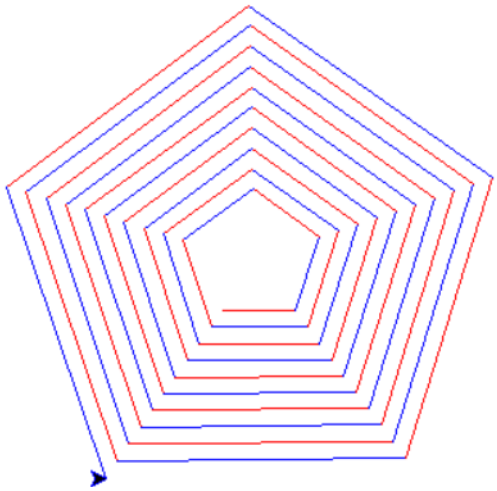


Рис. 2.5: Заданная траектория движения робота

Наиболее рациональный способ синтеза алгоритма заключается в том, чтобы в качестве отправной точки взять некий простейший вариант и постепенно добавлять в него команды на основе анализа условий задачи, постоянно проверяя его работу так, чтобы работа алгоритма становилась все более и более удовлетворяющей условиям задачи. При этом поиск синтаксических ошибок будет достаточно прост, так как будет облегчен процесс их локализации — они будет располагаться во вновь добавленном коде. Именно так природа создавала все живущие на Земле биологические виды — вначале появились простейшие одноклеточные организмы, постепенно они усложнялись в процессе эволюции, пока не появился человек. Для алгоритма движения школьного робота хорошим исходным вариантом является следующий:

Код 2.4: Синтез алгоритма движения по расходящейся спирали

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5
6 stop()
```

Далее, анализируя заданную траекторию движения можно сделать вывод, что в ее основе лежит некая пятиугольная структура — правильный пятиугольник, соответственно разумным будет добавить в синтезируемый алгоритм оператор цикла `while`, который сможет обеспечить движение робота вдоль 5 сторон пятиугольника (строки 5, 6 и 7):

Код 2.5: Синтез алгоритма движения по расходящейся спирали

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5 i=0
6 while i<5:
7     i=i+1
8 stop()
```

После успешного запуска программы и проверки ее работоспособности без синтаксических ошибок можно добавить в тело цикла команды движения вперед и левого поворота на какой либо угол (строки 7—10):

Код 2.6: Синтез алгоритма движения по расходящейся спирали

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5 i=0
6 while i<5:
7     rf(100)
8     st(1.5)
9     lt(0)
10    st(1)
11    i=i+1
12 stop()
```

Под управлением этой программы робот движется по траектории:

Рис. 2.6: Синтез алгоритма движения по расходящейся спирали

Далее следует подобрать время разворота (строка 10) таким образом, чтобы получился пятиугольник:

Код 2.7: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5  i=0
6  while i<5:
7      rf(100)
8      st(1.5)
9      lt(0)
10     st(0.84)
11     i=i+1
12 stop()
```

В результате получится траектория:

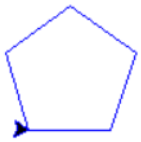


Рис. 2.7: Синтез алгоритма движения по расходящейся спирали

Анализируя заданную траекторию можно заметить, что в спирали каждая следующая сторона движения чуть больше предыдущей, это можно обеспечить применением арифметического выражения в 8 строке для последовательного увеличения времени движения вдоль очередной стороны, а значит и увеличения ее длины:

Код 2.8: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5  i=0
6  while i<5:
7      rf(100)
8      st(1.5+0.05*i)
9      lt(0)
10     st(0.84)
11     i=i+1
12 stop()
```

Под управлением этого алгоритма робот будет двигаться по траектории:

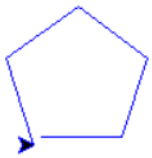


Рис. 2.8: Синтез алгоритма движения по расходящейся спирали

Для формирования спирали необходимо добавить количество повторений тела цикла (строка 6):

Код 2.9: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5  i=0
6  while i<50:
7      rf(100)
8      st(1.5+0.05*i)
9      lt(0)
10     st(0.84)
11     i=i+1
12 stop()
```

При этом робот будет двигаться по траектории:

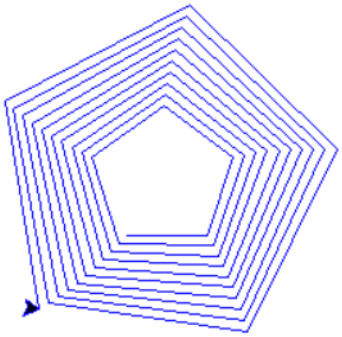


Рис. 2.9: Синтез алгоритма движения по расходящейся спирали

Для докручивания спирали следует уточнить время разворота робота (строка 10):

Код 2.10: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5  i=0
6  while i<50:
7      rf(100)
8      st(1.5+0.05*i)
9      lt(0)
10     st(0.8423)
11     i=i+1
12 stop()
```

Это приведет к следующей траектории движения робота:

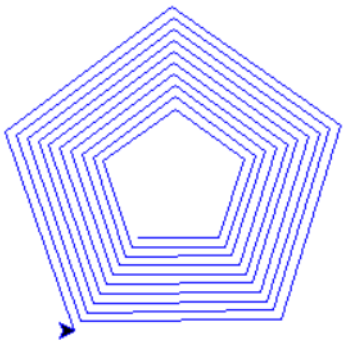


Рис. 2.10: Синтез алгоритма движения по расходящейся спирали

Теперь можно добавить управление светодиодом (строка 7):

Код 2.11: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5  i=0
6  while i<50:
7      LED.value(1-LED.value())
8      rf(100)
9      st(1.5+0.05*i)
10     lt(0)
11     st(0.8423)
12     i=i+1
13 stop()
```

И получить итоговую траекторию:

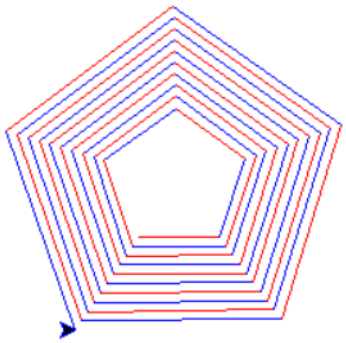


Рис. 2.11: Синтез алгоритма движения по расходящейся спирали

Таким образом, взяв в качестве исходной точки простейший алгоритм и последовательно уточняя его, шаг за шагом добавляя и корректируя команды, можно синтезировать такой алгоритм, который будет решать поставленную задачу с приемлемым качеством.

Следует отметить, что при попытке отладить тело цикла с помощью инструмента отладки “Перепрыгнуть”:

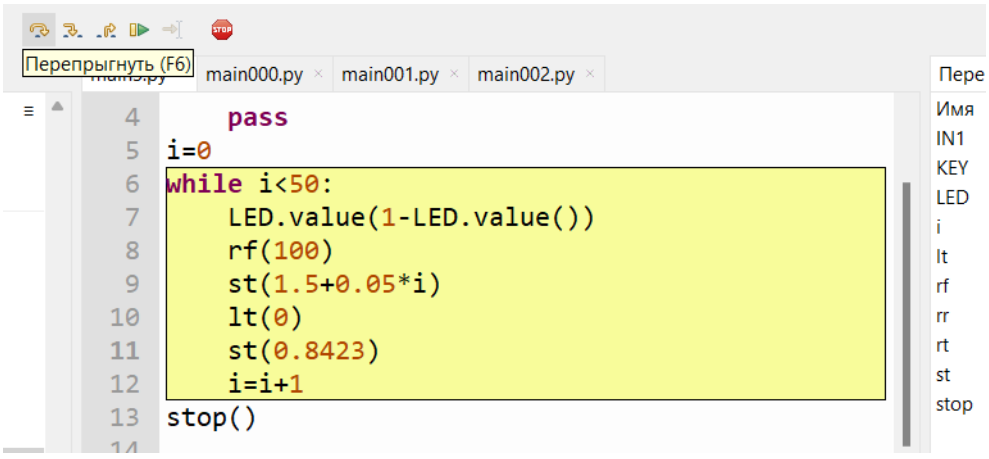


Рис. 2.12: Отладка тела цикла

цикл будет выполнен целиком и первым в очереди на выполнение станет следующий за циклом оператор:

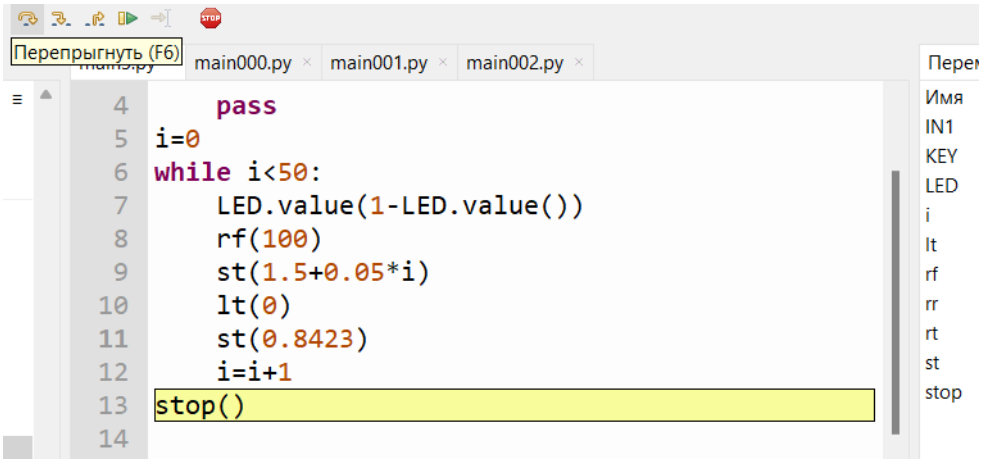


Рис. 2.13: Отладка тела цикла

Для того чтобы отлаживать операторы тела цикла можно воспользоваться отладочным инструментом “Зайти”:

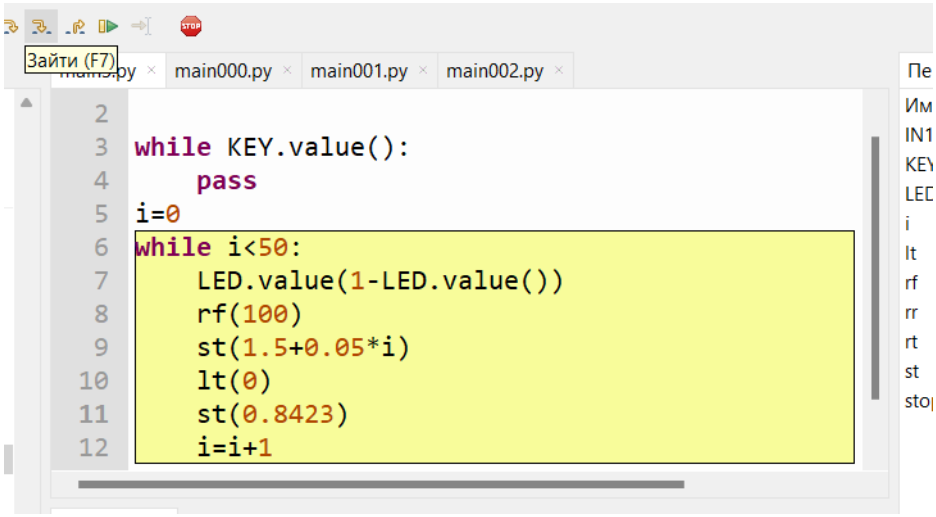


Рис. 2.14: Отладка тела цикла

После этого можно будет пользоваться инструментом “Перепрыгнуть” в обычном порядке:

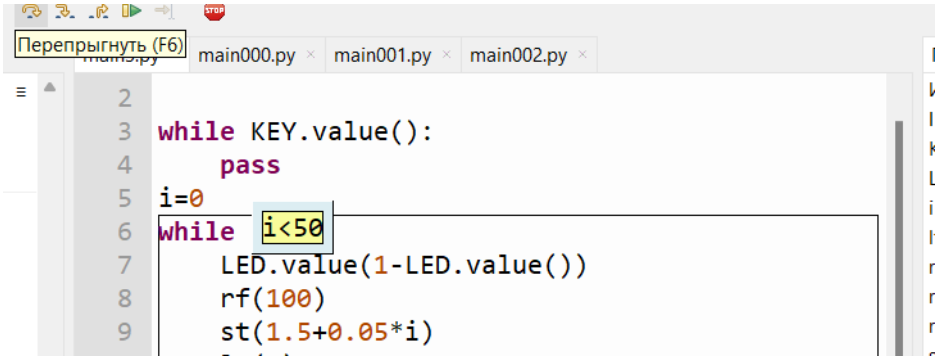


Рис. 2.15: Отладка тела цикла

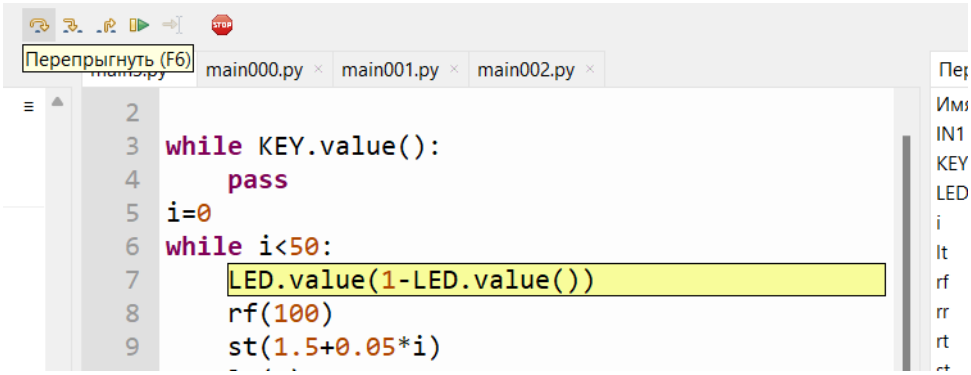


Рис. 2.16: Отладка тела цикла

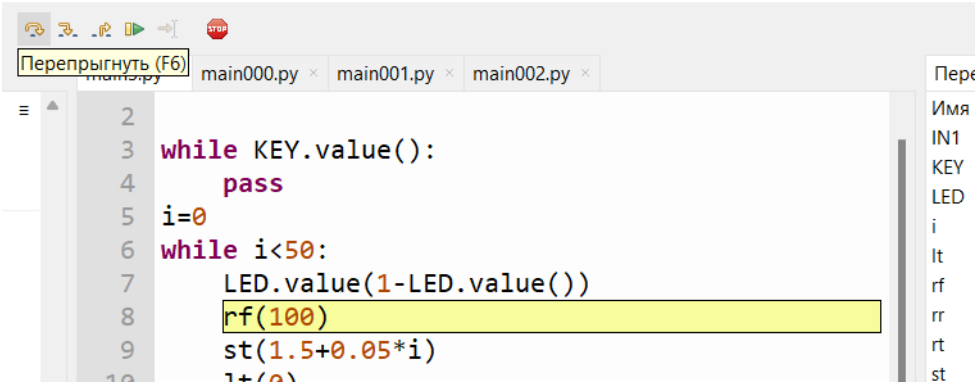


Рис. 2.17: Отладка тела цикла

Закончить отладку программы можно продолжив ее выполнение с помощью инструмента “Продолжить”:

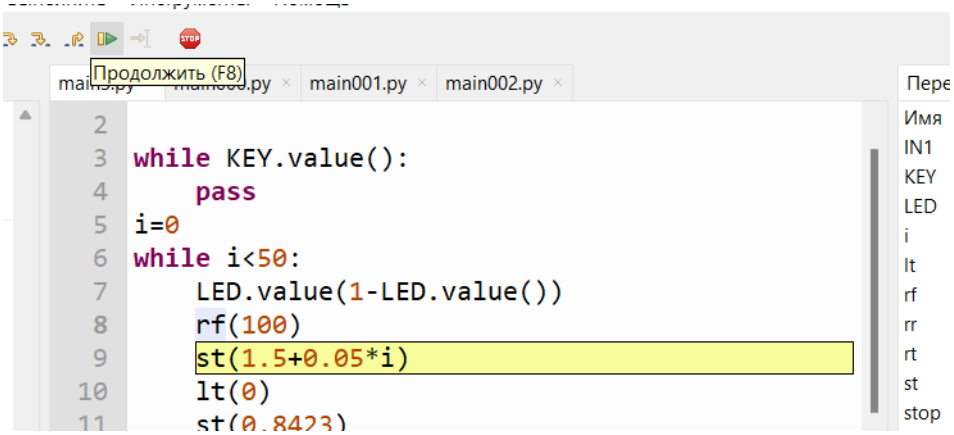


Рис. 2.18: Продолжение выполнения программы

Задания

1. Ознакомьтесь в каком либо руководстве по Python с арифметическими операциями и операциями сравнения. Изучите приоритет их выполнения. Обратите внимание: в англоязычной среде эти операции называются *операторами*.

2. Ознакомьтесь в каком либо руководстве по Python с правилами вычисления выражений.

3. Запрограммируйте робота, используя код 2.2. Объясните поведение робота.

4. Запрограммируйте робота для движения по квадрату, используя цикл `while`. Зажигайте светодиод на четных сторонах. Составьте блок-схему.

5. Запрограммируйте робота для движения по правильному пятиугольнику, используя цикл `while`. Зажигайте светодиод на четных сторонах. Составьте блок-схему.

6. Запрограммируйте робота для движения по звезде, используя цикл `while`. Зажигайте светодиод на четных сторонах. Составьте блок-схему.

2.3 Цикл for

Наряду с циклом `while` в Python существует еще один вид цикла - `for`. С применением этого цикла код 2.2 будет выглядеть следующим образом:

Код 2.12: Использование цикла `for`

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3
4 while KEY.value():
5     pass
6 g=0.5
```

```
for i in [1, 2, 3]:
    LED.value(0)
    rf(100)
    st(1.5)
    rt(0)
    st(g)
    LED.value(1)
    rf(100)
    st(1.5)
    lt(0)
    st(g)
stop()
```

Соответствующая блок схема представлена на рисунке 2.43

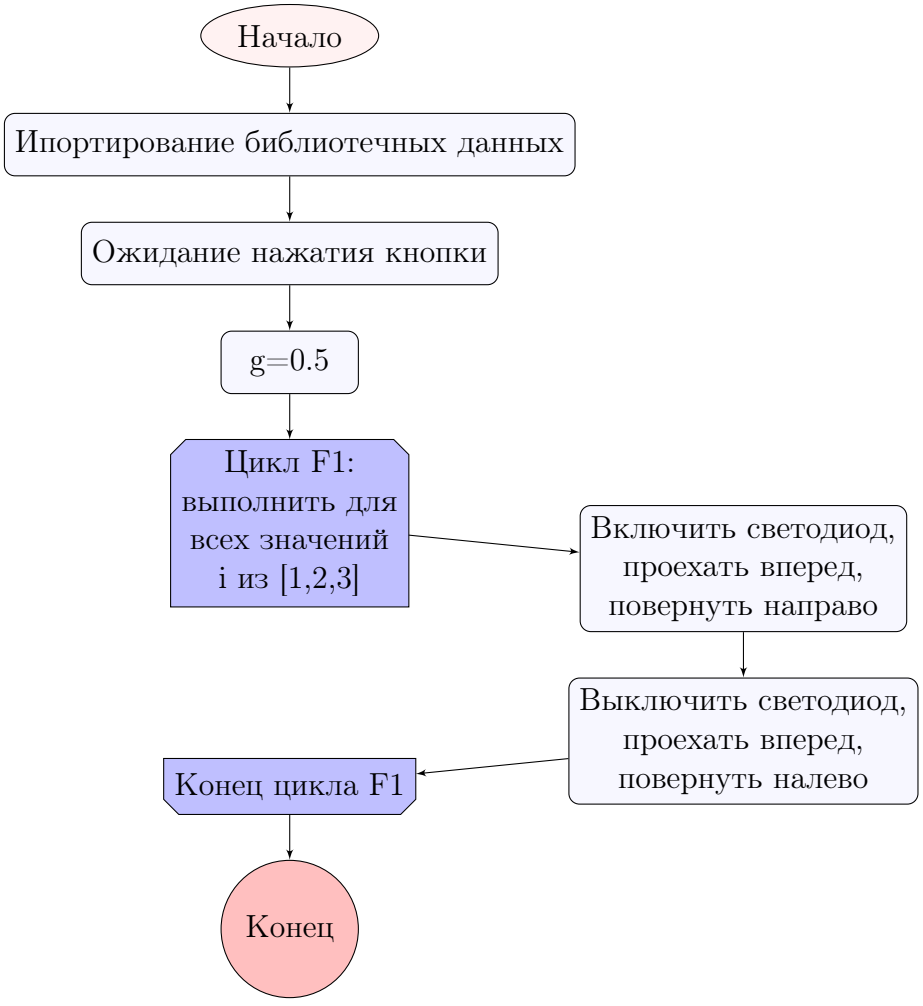


Рис. 2.19: Блок-схема с циклом for

Оператор цикла `for` схематически может быть представлен в виде алгоритмического блока:

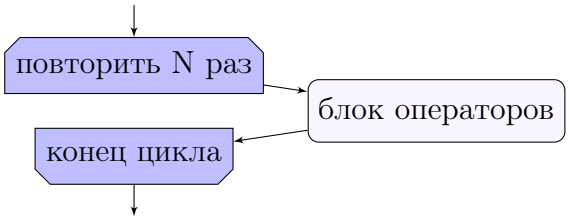


Рис. 2.20: Алгоритмический блок for

Структура оператора цикла `for` имеет следующий вид:

```
for параметр in набор значений:
    блок операторов
```

Цикл `for` называется циклом с заданным количеством повторений. Цикл выполняет `блок операторов` для каждого значения `параметра` из `набора значений`. Так, в предыдущем коде 2.12 `блок операторов` повторяется три раза - для каждого значения переменной `i`, принимающей поочередно значения: 1, 2, 3.

В качестве `набора значений` могут выступать *списки, кортежи, строки*, создаваемая функцией `range()` последовательность чисел.

В коде коде 2.12 в качестве набора значений выступает список

```
[1, 2, 3],
```

значения элементов которого последовательно присваиваются переменной `i`. Списки записываются в виде перечисления элементов через запятую и заключенных в квадратные скобки.

Строка 7 того же самого цикла с использованием кортежа будет выглядеть следующим образом:

```
for i in (1,2,3):
```

При использовании строк эту строку можно записать следующим образом:

```
for st in '123':
```

И последний вариант записи:

```
for i in range(1,4):
```

Относительно функции `range()` необходимо сказать несколько слов. Результатом ее работы являются целые числа. Ее параметрами тоже выступают целые числа. Например, `range(1,4)` генерирует последовательность 1, 2, 3. Последовательность для `range(a,b)` строится по следующему правилу: количество чисел в последовательности равно разности `b-a`, первое число в последовательности равно `a`, шаг последовательности равен 1, число `b` в последовательность не входит. Еще пример: `range(-1,4)` генерирует -1, 0, 1, 2, 3 — всего 5 чисел. Более подробно о функции `range()` можно почитать в каком-либо учебнике по языку Python.

В качестве примера применения цикла `for` ниже рассмотрена задача синтеза алгоритма движения робота по расходящейся спирали:

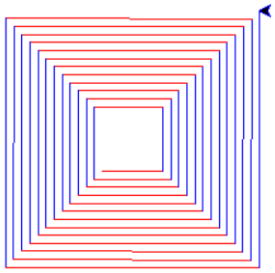


Рис. 2.21: Заданная траектория движения робота

Как и в предыдущем параграфе синтез алгоритма будет выполняться по шагам и хорошим исходным вариантом алгоритма является следующий:

Код 2.13: Синтез алгоритма движения по расходящейся спирали

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5
6 stop()
```

Далее, анализируя заданную траекторию движения можно сделать вывод, что в ее основе лежит четырехугольная структура — квадрат, соответственно разумным будет добавить в синтезируемый алгоритм оператор цикла `for`, который сможет обеспечить движение робота вдоль 4 сторон квадрата (строки 6 и 7):

Код 2.14: Синтез алгоритма движения по расходящейся спирали

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5
6 for i in range(1,5):
7     pass
8 stop()
```

После успешного запуска программы и проверки ее работоспособности без синтаксических ошибок можно добавить в тело цикла команды движения вперед и левого поворота на какой либо угол (строки 7—10):

Код 2.15: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(1,5):
7      rf(100)
8      st(1.5)
9      lt(0)
10     st(1)
11 stop()
```

Под управлением этой программы робот движется по траектории:

Рис. 2.22: Синтез алгоритма движения по расходящейся спирали

Далее следует подобрать время разворота (строка 10) таким образом, чтобы получился квадрат:

Код 2.16: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(1,5):
7      rf(100)
8      st(1.5)
9      lt(0)
10     st(1.05)
11 stop()
```

В результате получится траектория:

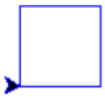


Рис. 2.23: Синтез алгоритма движения по расходящейся спирали

Анализируя заданную траекторию можно заметить, что в спирали каждая следующая сторона движения чуть больше предыдущей, это можно обеспечить применением арифметического выражения в 8 строке для последовательного увеличения времени движения вдоль очередной стороны, а значит и увеличения ее длины:

Код 2.17: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(1,5):
7      rf(100)
8      st(1.5+0.1*i)
9      lt(0)
10     st(1.05)
11 stop()
```

Под управлением этого алгоритма робот будет двигаться по траектории:

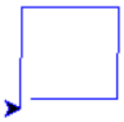


Рис. 2.24: Синтез алгоритма движения по расходящейся спирали

Для формирования спирали необходимо добавить количество повторений тела цикла (строка 6):

Код 2.18: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(1,51):
7      rf(100)
8      st(1.5+0.1*i)
9      lt(0)
10     st(1.05)
11 stop()
```

При этом робот будет двигаться по траектории:

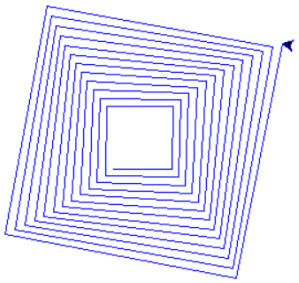


Рис. 2.25: Синтез алгоритма движения по расходящейся спирали

Для докручивания спирали следует уточнить время разворота робота (строка 10):

Код 2.19: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(1,51):
7      rf(100)
8      st(1.5+0.1*i)
9      lt(0)
10     st(1.0526)
11 stop()
```

Это приведет к следующей траектории движения робота:

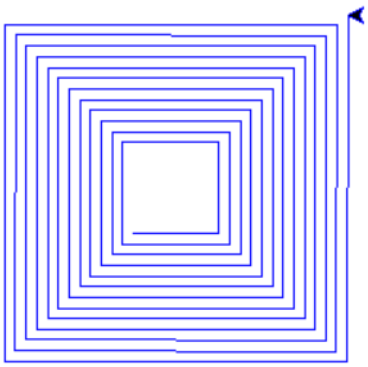


Рис. 2.26: Синтез алгоритма движения по расходящейся спирали

Теперь можно добавить управление светодиодом (строка 7):

Код 2.20: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(1,51):
7      LED.value(1-LED.value())
8      rf(100)
9      st(1.5+0.1*i)
10     lt(0)
11     st(1.0526)
12 stop()
```

И получить итоговую траекторию:

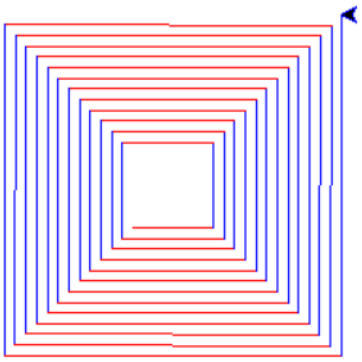


Рис. 2.27: Синтез алгоритма движения по расходящейся спирали

Задания

1. Ознакомьтесь в каком либо руководстве по Python с типами данных: списками, кортежами, строками, а также с использованием функции `range()`.

2. Перепишите код 2.12 с использованием кортежа. Нарисуйте соответствующую блок-схему.

3. Перепишите код 2.12 с использованием строки. Нарисуйте соответствующую блок-схему.

4. Перепишите код 2.12 с использованием функции `range()`. Нарисуйте соответствующую блок-схему.

2.4 Использование циклов, движение по многоугольникам и спирали

Использование циклов позволяет существенно упростить программу. Ниже представлен код для движения по треугольнику:

Код 2.21: Движение по треугольнику

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
g=1.4
for i in range(1,4):
    LED.value(1-LED.value())
    rf(100)
    st(1.5)
    rt(0)
    st(g)
LED.value(1)
stop()
```

Более короткую программу отлаживать гораздо проще, чем длинную. Поэтому всегда нужно пытаться найти в алгоритме повторяющиеся блоки команд и использовать алгоритмическую конструкцию повторение.

Ниже приведен еще один пример кода с циклом - код для движения по спирали:

Код 2.22: Движение по спирали

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
g=1.05
for i in range(1,10):
    LED.value(1-LED.value())
    rf(100)
    st(i)
    rt(0)
    st(g)
LED.value(1)
stop()
```

Цикл `for` может быть легко преобразован в цикл `while`. Например код 2.21 может быть переписан в виде:

Код 2.23: Движение по треугольнику

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
g=1.4
i=1
while i<4:
    LED.value(1-LED.value())
    rf(100)
    st(1.5)
    rt(0)
    st(g)
    i=i+1
LED.value(1)
stop()
```

а код 2.22 — в виде:

Код 2.24: Движение по спирали

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
g=1.05
i=1
while i<10:
    LED.value(1-LED.value())
```

```
10      rf(100)
11      st(i)
12      rt(0)
13      st(g)
14      i=i+1
15  LED.value(1)
16  stop()
```

Задания

1. Нарисуйте блок-схему алгоритма движения по треугольнику (код 2.21).
2. Нарисуйте блок-схему алгоритма движения по спирали (код 2.22). Объясните за счет чего получается расширяющаяся спираль.
3. Запрограммируйте робота, используя код 2.21. Добейтесь его движения по треугольнику.
4. Запрограммируйте робота для движения по спирали, используя цикл `while`. Нарисуйте блок схему алгоритма.
5. Запрограммируйте робота для движения по правильному пятиугольнику, используя цикл `for` и кортежи. Нарисуйте блок схему алгоритма.

2.5 Ветвление

Наряду с алгоритмическими конструкциями следование и повторение существует еще одна алгоритмическая конструкция, существенно облегчающая создание алгоритмов - ветвление. Начать знакомиться с ней можно на примере операции деления. При попытке выполнить следующий код:

Код 2.25: Деление на 0

```
1  d=4
2  dv=0
3  f=d/dv
```

возникает ошибка деления на 0 и выполнение программы аварийно завершается:

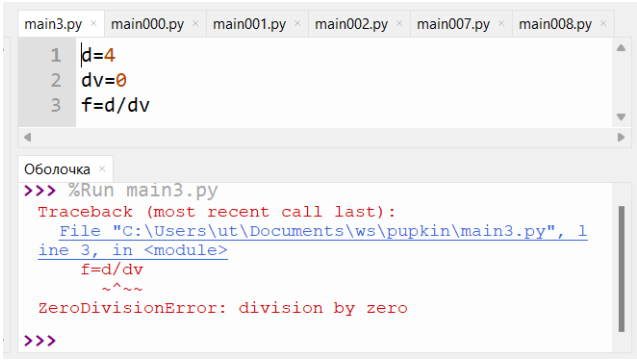


Рис. 2.28: Аварийное завершение программы с делением на 0

Такая ситуация может возникнуть при написании программы простейшего калькулятора, когда она получает из какого-либо источника числа и должна их поделить одно на другое. Если в качестве делителя будет получен ноль, то программа сломается. Вариантом решения этой проблемы является использование алгоритмической конструкции ветвление, которая на языке Python реализована с помощью условного оператора `if` (операция `!=` означает `не равно`):

Код 2.26: Использование условного оператора

```
d=4
dv=0
if  dv!=0:
     f=d/dv
```

выполнение этого кода происходит без ошибок:

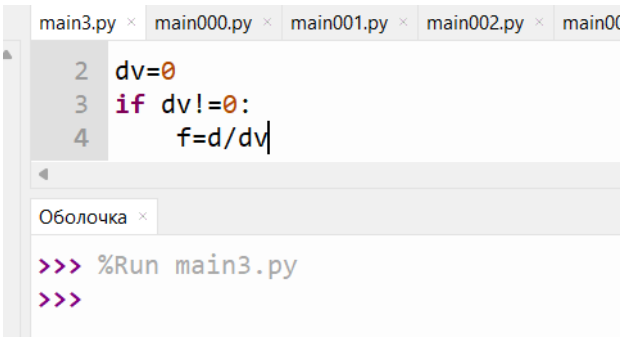


Рис. 2.29: Корректное завершение программы с условным оператором

В этом примере использован условный оператор в виде неполного ветвления. Соответствующий алгоритмический блок выглядит следующим образом:

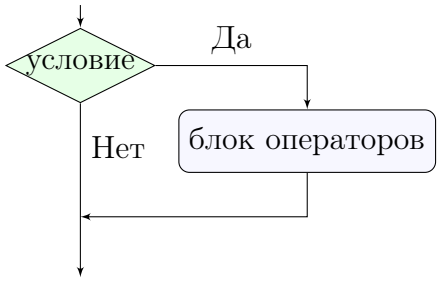


Рис. 2.30: Алгоритмический блок неполного ветвления

Блок-схема самой программы выглядит так:

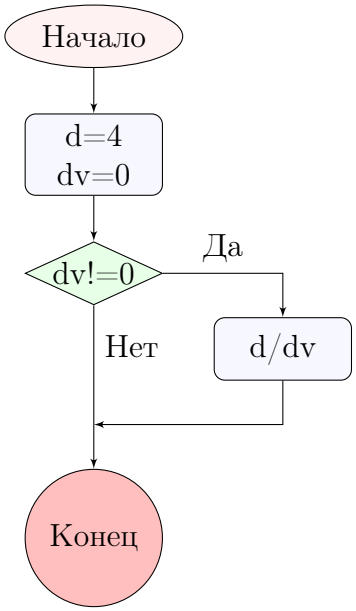


Рис. 2.31: Блок-схема программы с неполным ветвлением

Структура условного оператора с неполным ветвлением имеет вид:

```
if условие:
    блок операторов
```

`Блок операторов` выполняется только в том случае если `условие` истинно. Если `условие` ложно, то `блок операторов` не выполняется. Это ветвление называется неполным потому, что `блок операторов` может быть выполнен, а может и не выполняться — в зависимости от истинности или ложности условия.

В рассматриваемом примере операция деления `d/dv` выполняется, если истинно условие `значение переменной dv не равно 0`, в противном случае операция деления не выполняется, и ошибки деления на 0 не возникает.

Пример с калькулятором может использовать и другой вид условного оператора:

Код 2.27: Использование условного оператора

```
1 d=4
2 dv=0
3 if dv!=0:
4     f=d/dv
5 else:
6     print('на ноль делить нельзя')
```

Результат работы этой программы приведен на рисунке:

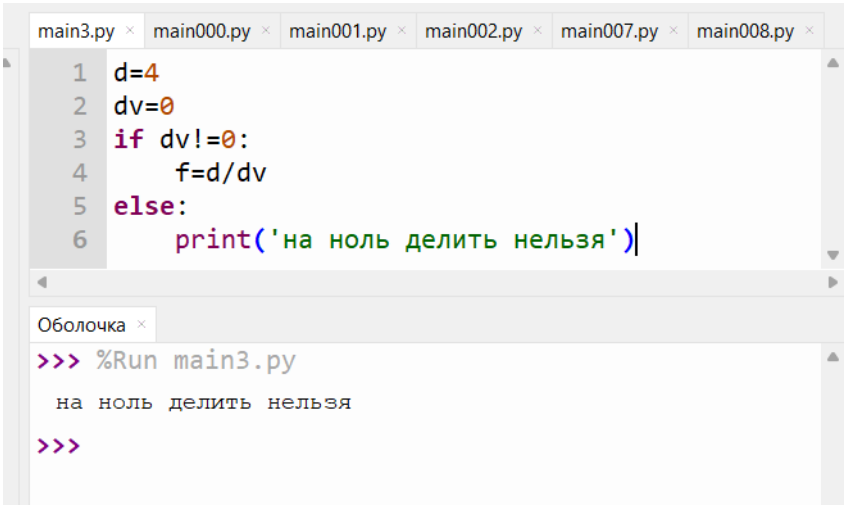


Рис. 2.32: Корректное завершение программы с условным оператором

В этом примере использован вариант так называемого полного ветвления. Помимо того, что программа завершилась без ошибок она еще вывела предупредительную надпись, что была попытка деления на 0.

В этой программе используется специальная функция `print()`. Она служит для вывода информации в окно панели “Оболочка”, и может выводить строки и значения переменных. Для вывода строки в скобках указывается сама строка: `print(‘‘строка для вывода’’)`. Строки заключаются в одинарные, либо двойные кавычки. Для того, чтобы вывести значение переменной в скобках указывается имя переменной: `print(dv)`. Также эта функция может выводить значение выражений: `print(d/dv)`. Кроме того эта функция может выводить несколько объектов, для этого объекты указываются в скобках через запятую: `print(‘‘d/dv=’’, d/dv)`. Более подробно о функции `print()` можно прочитать в каком-либо учебнике по языку Python.

Алгоритмический блок для условного оператора в виде полного ветвления выглядит следующим образом:

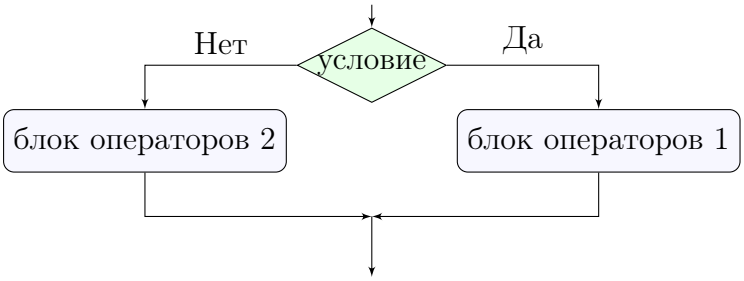


Рис. 2.33: Алгоритмический блок полного ветвления

Блок-схема самой программы выглядит так:

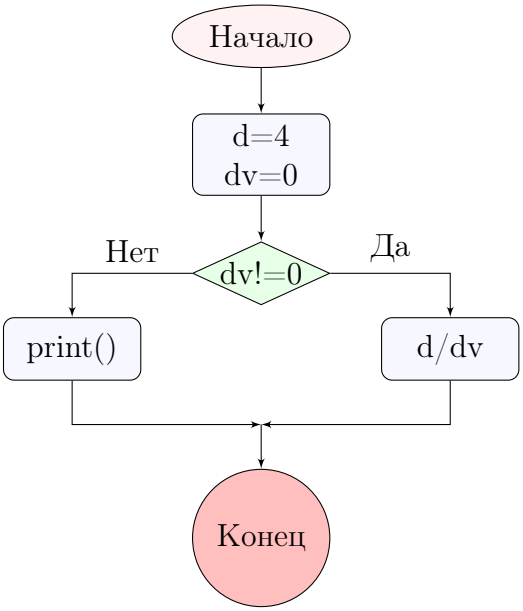


Рис. 2.34: Блок-схема программы с неполным ветвлением

Структура этого условного оператора имеет следующий вид:

```
if условие:
    блок операторов 1
else:
    блок операторов 2
```

Если `условие` истинно, то выполняется `блок операторов 1`, если `условие` ложно - выполняется `блок операторов 2`, т.е. происходит выбор одного из нескольких вариантов.

Рассмотренный вариант ветвления называется полным ветвлением, поскольку выполняется либо один блок операторов, либо другой.

Ход вычислительного процесса при использовании условного оператора можно пронаблюдать с помощью отладчика. Когда первым в очереди на выполнение становится условный оператор следует воспользоваться инструментом “Зайти”, чтобы попасть внутрь этого условного оператора:

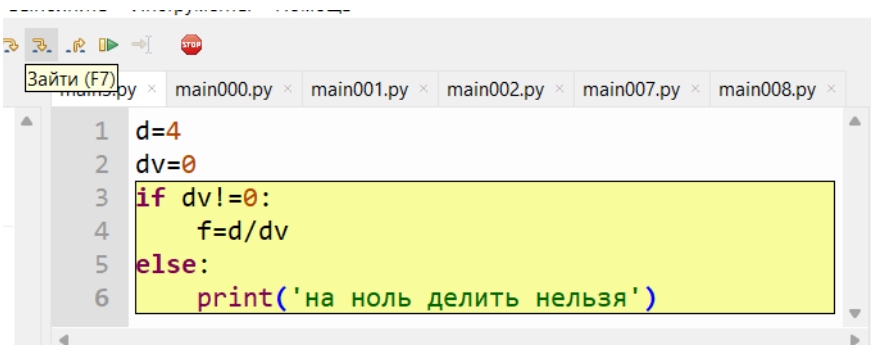


Рис. 2.35: Пошаговое прохождение ветвления

После применения этого инструмента первым в очереди на выполнение будет проверка условия. Так как в данном случае условие будет ложным, так как `dv=0`, то после применения инструмента “Перепрыгнуть”:

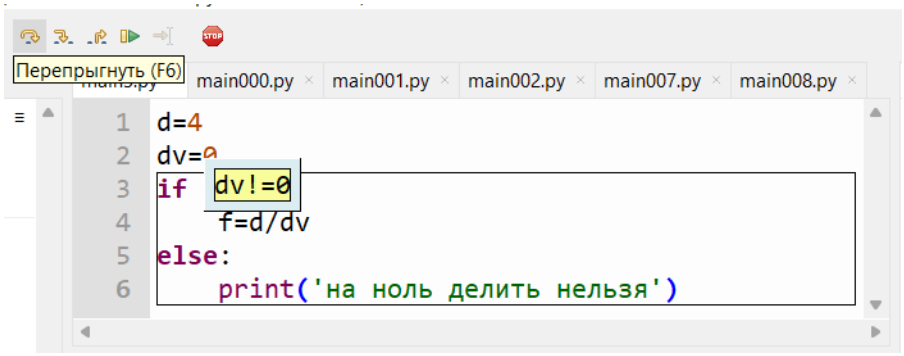


Рис. 2.36: Пошаговое прохождение ветвления

начнет выполняться ветвь “`else`”, содержащая вывод на печать сообщение о попытке деления на 0, а само деление выполняться не будет:

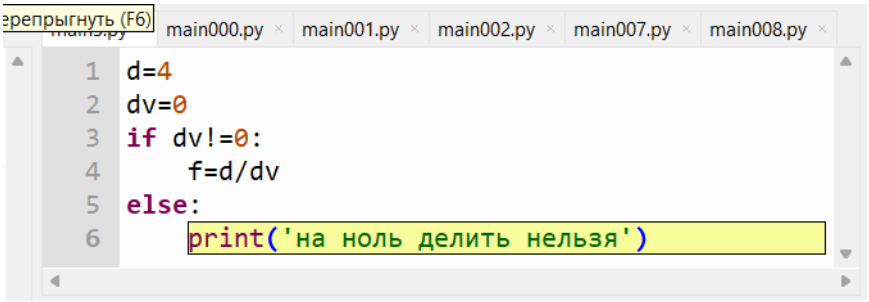


Рис. 2.37: Пошаговое прохождение ветвления

Если же значение переменной `dv` будет равно 1, то после применения инструмента “Зайти”:

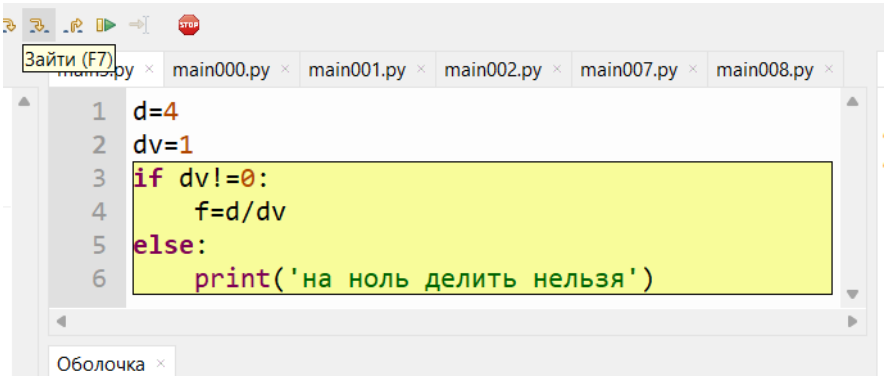


Рис. 2.38: Пошаговое прохождение ветвления

и проверки условия, которое будет истинным, поскольку единица действительно не равна нулю:

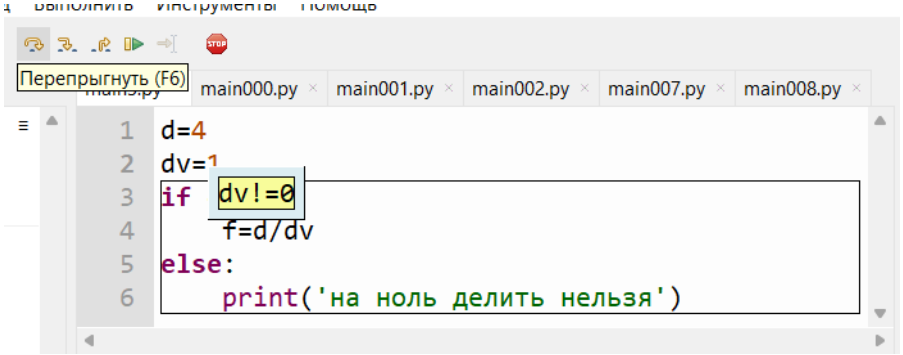


Рис. 2.39: Пошаговое прохождение ветвления

будет выполняться ветвь условного оператора с делением, а ветвь “`else`”, содержащая вывод на печать сообщение о попытке деления на 0, выполняться не будет:

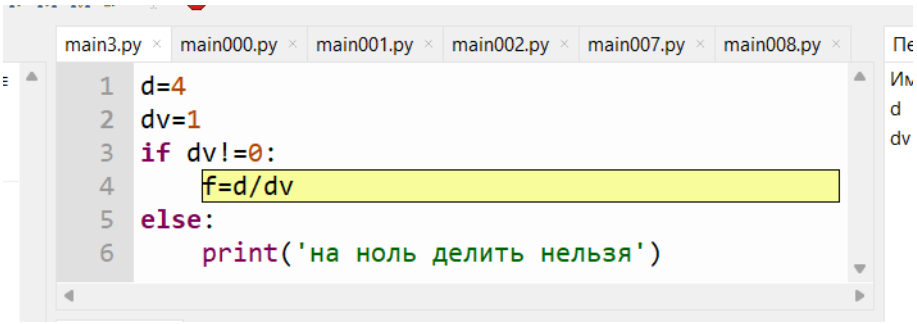


Рис. 2.40: Пошаговое прохождение ветвления

Все это легко проверить самостоятельно в IDE Olly.

Задания

1. Приведите примеры использования конструкции полного ветвления в реальной жизни.

2. Приведите примеры использования конструкции неполного ветвления в реальной жизни.

2.6 Вложенные конструкции, вложенные ветвления, `elif`

При построении алгоритмов следует стараться использовать только рассмотренные алгоритмические конструкции: следование, повторение и ветвление, и соответствующие им операторы циклов, а также условный оператор. При этом в языке Python возможно использовать 5 принципиально различных алгоритмических блоков:

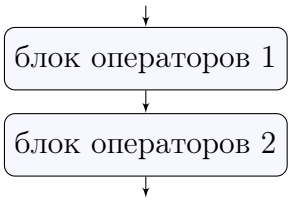


Рис. 2.41: Алгоритмический блок следование

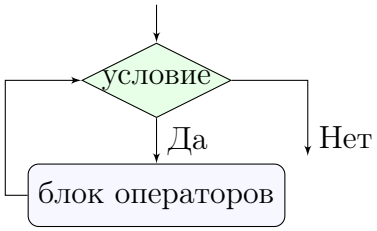


Рис. 2.42: Алгоритмический блок while

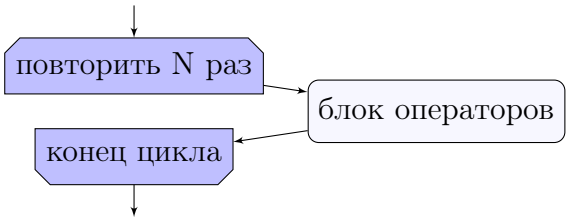


Рис. 2.43: Алгоритмический блок for

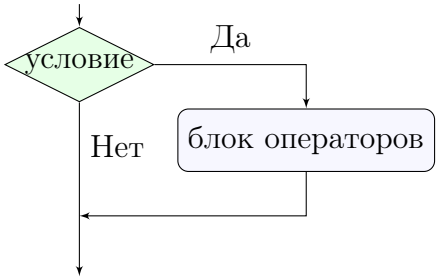


Рис. 2.44: Алгоритмический блок неполного ветвления

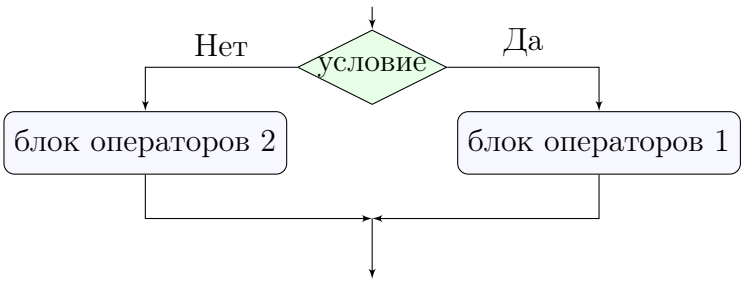


Рис. 2.45: Алгоритмический блок полного ветвления

Указанные алгоритмические конструкции могут применяться как независимо друг от друга, так и быть вложенными друг в друга. Так среди операторов в блоках операторов, представленных выше алгоритмических блоков, могут быть и циклы и ветвления. Это можно представить себе как матрешек внутри которых находятся другие матрешки, а внутри них еще матрешки и так далее. И внутри любого алгоритмического блока (конкретно — внутри прямоугольников с блоками операторов) могут располагаться любые другие алгоритмические блоки.

В качестве примера можно рассмотреть калькулятор, умеющий выполнять две операции — сложение и умножение:

Код 2.28: Вложенные ветвления

```
n1=4
n2=1
op='+'
if op=='+':
    print(n1+n2)
else:
    if op=='*':
        print(n1*n2)
    else:
        print('неизвестная операция')
```

В этом алгоритме используются два ветвления, одно вложенное в другое. В этом примере вначале проверяется условие, что значение переменной `op` равно строке `'+'`, если оно истинно, то программа выводит результат сложения двух чисел. Если же переменная `op` содержит какое то другое значение, то проверяется новое условие: что значение переменной `op` равно строке `'*'`. Если это новое условие истинно, то программа выводит произведение двух чисел, если же это условие ложно, то программа сообщает, что встретила неизвестную операцию. Получается, что один условный оператор работает внутри другого условного оператора.

В языке Python имеется конструкция, которая позволяет более красиво работать с вложенными условными операторами и формально сворачивать их в один условный оператор:

```
if условие1:
    блок операторов 1
elif условие2:
    блок операторов 2
else:
    блок операторов 3
```

Логика работы этой конструкции следующая: вначале проверяется `условие1`, если оно истинно, то выполняется `блок операторов 1`; если первое условие ложно, то проверяется `условие2`, если истинно оно, то выполняется `блок операторов 2`; если оба условия ложны, то выполняется `блок операторов 3`.

С применением этой конструкции предыдущий код может быть переписан в виде:

Код 2.29: Вложенные ветвления

```
n1=4
n2=1
op='*'
if  op=='+':
     print(n1+n2)
elif  op=='*':
     print(n1*n2)
else:
     print('неизвестная операция')
```

При этом блок-схема для обоих реализаций алгоритма одна и та же:

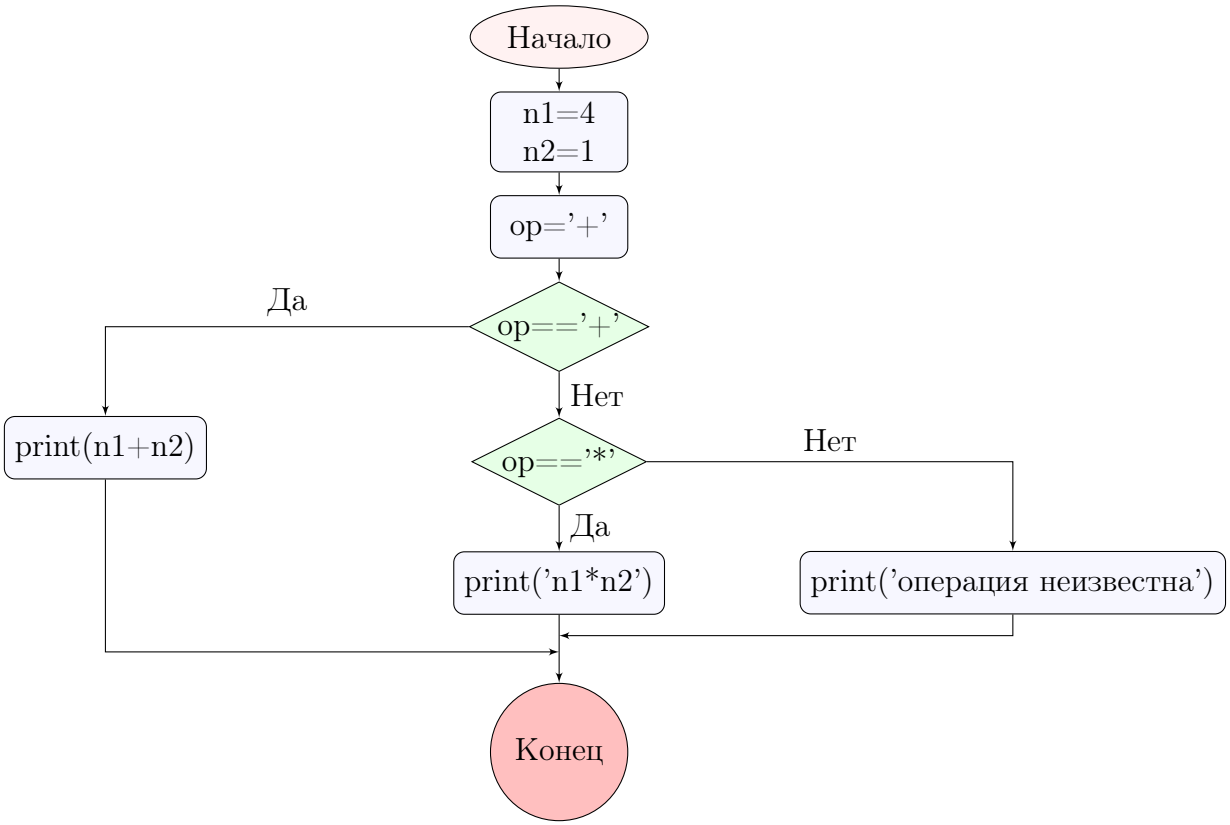


Рис. 2.46: Блок-схема вложенных ветвлений

По субъективным ощущениям автора использование условного оператора `if` для множественного ветвления с применением конструкции `elif` интуитивно более понятно.

Задания

1. Приведите примеры использования вложенных ветвлений в реальной жизни.

2. Доработайте программу этого параграфа, чтобы она умела еще делить и вычитать. Нарисуйте для нее блок-схему.

2.7 Ветвление, вложенное в цикл

Широко распространена схема, в которой внутри цикла происходит ветвление.

Если внимательно посмотреть на код 2.2, можно увидеть, что внутри цикла два раза повторяется почти один и тот же код. Разница в том, что в первом фрагменте выполняется правый поворот, а во втором фрагменте - левый, т.е, фактически, внутри цикла происходит выбор одного из двух вариантов - повернуть направо или налево.

Чтобы понять как можно сократить код 2.2 имеет смысл нарисовать схему движения робота (рисунок 2.48).

Рис. 2.47: Схема движения робота

Код 2.2

Отрезки со стрелочками обозначают участки движения робота вперед. Красный цвет обозначает участки движения где светодиод включен, синий - выключен. Цифра над отрезками - значение счетчика цикла. При сокращении кода внутри цикла робот будет проходить только один отрезок, следовательно, чтобы робот прошел по той же траектории необходимо, чтобы количество повторений выросло с трех до шести раз. Соответственно, значение счетчика цикла будет меняться чаще - на каждом прямолинейном отрезке движения робота, и соответственно это значение можно считать номером прямолинейного отрезка движения робота:

Рис. 2.48: Схема движения робота

Видно, что участки включенного и выключенного светодиода чередуются, поэтому для управления светодиодом можно использовать команду `LED.value(1-LED.value())`. Для выбора стороны поворота в конце прямолинейных участков можно использовать условный оператор. Таким образом код 2.2 можно переписать в следующем виде:

Код 2.30: Ветвление, вложенное в цикл

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3
4
5 while KEY.value():
6     pass
7 g=0.5
8 i=0
```

```
while i<6:
    LED.value(1-LED.value())
    rf(100)
    st(1.5)
    if (i%2)==0:
        rt(0)
        st(g)
    else:
        lt(0)
        st(g)
    i=i+1
stop()
```

Соответствующая блок-схема:

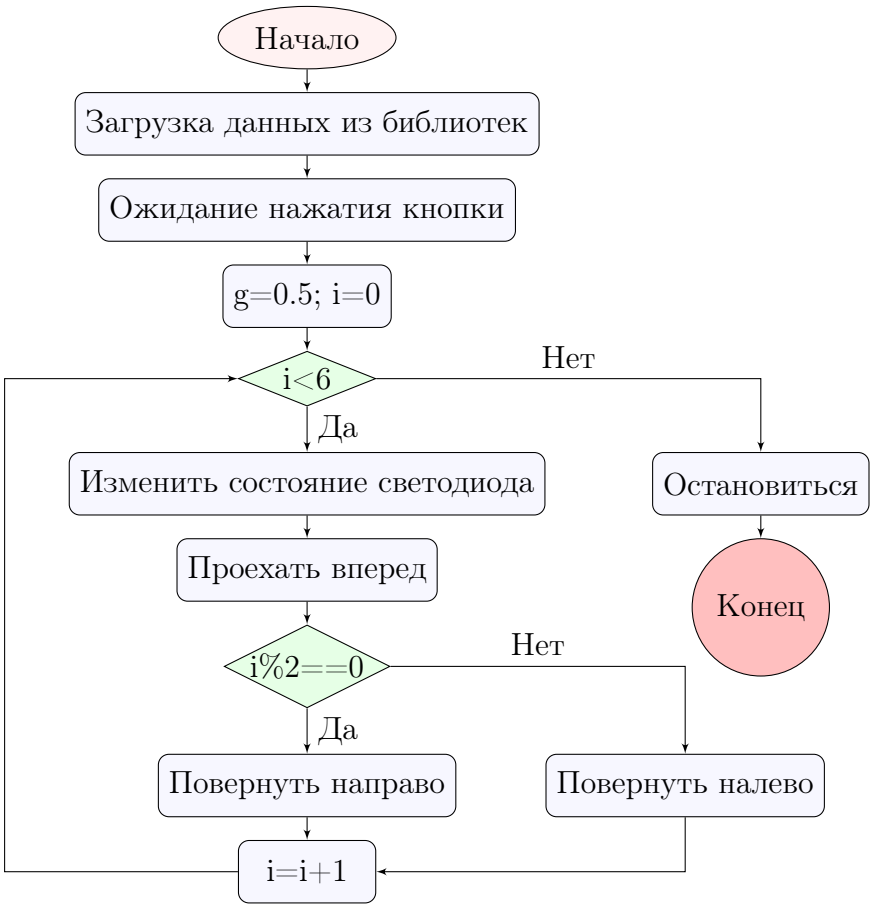


Рис. 2.49: Блок-схема ветвления вложенного в цикл

Операция % обозначает взятие остатка от деления, т.е. робот поворачивает направо на тех отрезках, для которых остаток от деления значения счетчика цикла на два равен нулю и - налево во всех остальных случаях.

Следует отметить, что данная комбинация алгоритмических конструкций - ветвление в цикле имеет огромное практическое значение. Она используется во многих системах, где есть внешний рабочий цикл, внутри которого происходит ветвление - обработка внешних событий. Например, любой смартфон работает по этой схеме.

В качестве еще одного примера ветвления, вложенного в цикл ниже рассмотрена задача синтеза алгоритма движения робота по схеме:

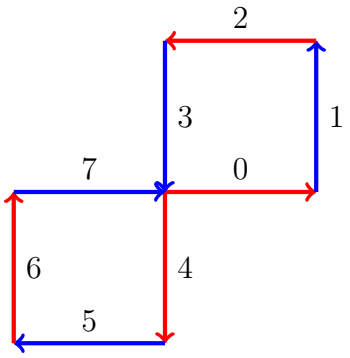


Рис. 2.50: Схема движения робота

Как и ранее синтез алгоритма будет выполняться по шагам и хорошим исходным вариантом алгоритма является следующий:

Код 2.31: Синтез алгоритма движения робота

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5
6 stop()
```

Далее, анализируя заданную траекторию движения можно сделать вывод, что робот должен пройти 8 отрезков, соответственно разумным будет добавить в синтезируемый алгоритм оператор цикла `for`, который сможет обеспечить движение робота вдоль 8 отрезков (строки 6 и 7):

Код 2.32: Синтез алгоритма движения робота

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5
6 for i in range(0,8):
7     pass
8 stop()
```

После успешного запуска программы и проверки ее работоспособности без синтаксических ошибок можно добавить в тело цикла команды движения вперед и левого поворота на угол 90°(строки 7—10):

Код 2.33: Синтез алгоритма движения робота

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

while KEY.value():
    pass

for i in range(0,8):
    rf(100)
    st(1.5)
    lt(0)
    st(1.05)
stop()
```

Под управлением этой программы робот 2 раза проезжает по одному и тому же квадрату:

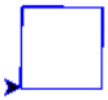


Рис. 2.51: Синтез алгоритма движения робота

Это происходит потому, что робот постоянно делает левый поворот после прямолинейного участка движения. Анализируя заданную траекторию можно заметить, что левый поворот робот должен делать только после 0, 1 и 2 участка движения, то есть для значений переменной `i<3`, это можно обеспечить применением условного оператора внутри цикла:

Код 2.34: Синтез алгоритма движения робота

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

while KEY.value():
    pass

for i in range(0,8):
    rf(100)
    st(1.5)
    if i<3:
        lt(0)
        st(1.05)
stop()
```

Под управлением этого алгоритма робот проезжает по траектории:

Рис. 2.52: Синтез алгоритма движения робота

Для формирования второго квадрата необходимо модернизировать условный оператор так, чтобы робот делал правый поворот после 4, 5 и 6 участка движения, то есть при значении счетчика повторений тела цикла `i>3`:

Код 2.35: Синтез алгоритма движения робота

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

while KEY.value():
    pass

for i in range(0,8):
    rf(100)
    st(1.5)
    if i<3:
        lt(0)
        st(1.05)
    elif i>3:
        rt(0)
        st(1.05)
stop()
```

При этом робот будет двигаться по траектории:

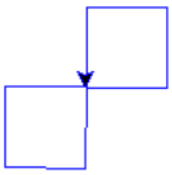


Рис. 2.53: Синтез алгоритма движения робота

Теперь можно добавить управление светодиодом (строка 7):

Код 2.36: Синтез алгоритма движения робота

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(0,8):
7      LED.value(1-LED.value())
8      rf(100)
9      st(1.5)
10     if i<3:
11         lt(0)
12         st(1.05)
13     elif i>3:
14         rt(0)
15         st(1.05)
16 stop()
```

И получить итоговую траекторию:

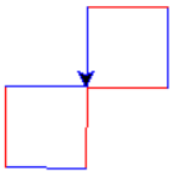


Рис. 2.54: Синтез алгоритма движения робота

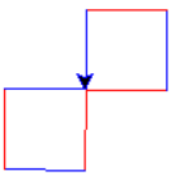
Задания

1. Приведите примеры использования ветвления, вложенного в цикл в реальной жизни.

2. Запрограммируйте робота, используя код 2.30. Объясните поведение робота.

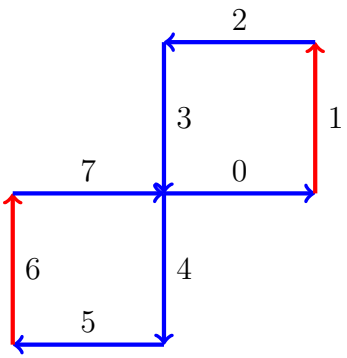
3. Перепишите код 2.30, используя цикл `for`. Запрограммируйте робота, используя полученный код. Объясните поведение робота. Нарисуйте блок-схему.

4. Напишите программу движения робота по траектории:



Но такую, чтобы робот не делал поворота после последнего участка прямолинейного движения.

5. Напишите программу движения робота по схеме:



2.8 Логические выражения

В предыдущих примерах использовались простые условия. При программировании реальных объектов приходится использовать более сложные условия с применением логических выражений на основе логических операций `and, or, not`. Эти операции имеют следующие определения:

1. Условие (`not условие1`) истинно в том и только том случае если `условие1` ложно.

2. Условие (`условие1 and условие2`) истинно в том и только том случае если `условие1` и `условие2` истинны одновременно.

3. Условие (`условие1 or условие2`) ложно в том и только том случае если `условие1` и `условие2` ложны одновременно.

В логическом выражении может одновременно присутствовать несколько логических операций, при этом они выполняются слева направо, с учетом их приоритета. Самый высокий приоритет у операции `not`, самый низкий - `or`. Скобки меняют приоритет - операции в скобках выполняются в первую очередь.

Следующий пример иллюстрирует применение логических выражений.

Пусть необходимо, чтобы робот проехал по квадратной восьмерке:

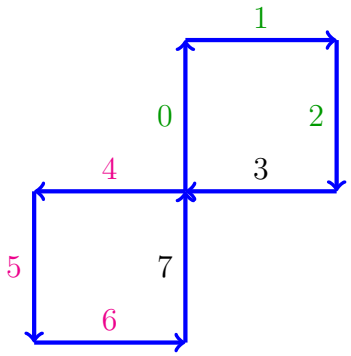


Рис. 2.55: Использование логических выражений

На этой схеме дополнительные обозначения: для отрезков, после которых следует поворот направо, значения счетчика цикла отображены темно-зеленым цветом, а

после которых следует поворот налево - малиновым. Черным цветом отображены значения счетчика для сторон после которых робот не поворачивает. Следующий код использует логические выражения для реализации такого поведения робота:

Код 2.37: Использование логических выражений

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
r=1.05
k=1.05
i=0
while i<8:
    rf(100)
    st(1.5)
    if i<3:
        rt(0)
        st(r)
    elif (i>3)and(i<7):
        lt(0)
        st(k)
    i=i+1
stop()
```

Блок-схема этого алгоритма приведена на рисунке 2.56:

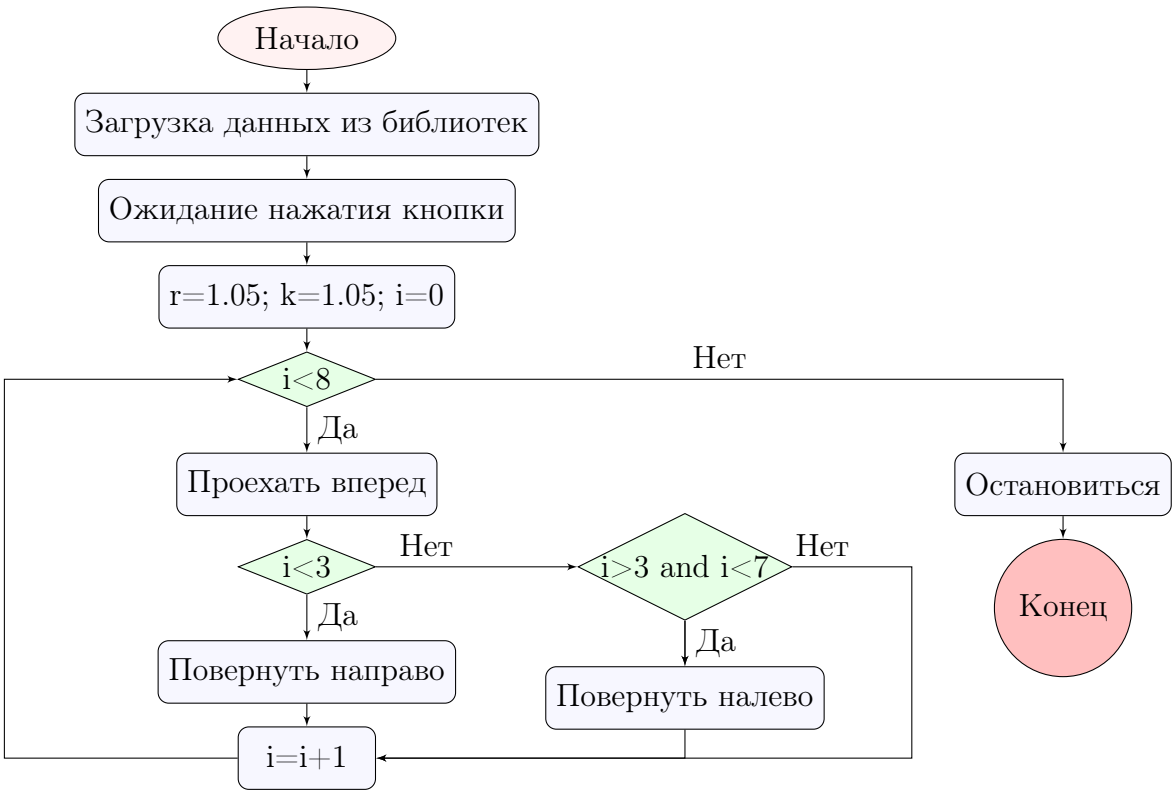


Рис. 2.56: Блок-схема движения по квадратной восьмерке

Следует отметить, что в этом программном коде повороты налево и направо настраиваются с помощью двух разных переменных по причине того, что у моторов реального робота могут быть слегка разные характеристики и повороты в разные стороны робот может осуществлять по разному. Строка 15 в коде 2.37 использует логическую операцию, которая обеспечивает то, что поворот налево будет выполняться только для значений параметра цикла, которые одновременно больше трех и меньше семи, то есть после последнего отрезка движения робот поворачиваться не будет.

Для более подробного рассмотрения логических операций ниже проведен синтез алгоритма движения робота по расходящейся восьмиугольной спирали:

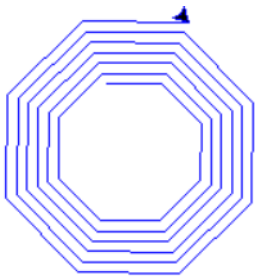


Рис. 2.57: Расходящаяся восьмиугольная спираль

Как и в предыдущем параграфе синтез алгоритма будет выполняться по шагам и хорошим исходным вариантом алгоритма является следующий:

Код 2.38: Синтез алгоритма движения по расходящейся спирали

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5
6 stop()
```

Далее, анализируя заданную траекторию движения можно сделать вывод, что в ее основе лежит некая восьмиугольная структура, соответственно, разумным будет добавить в синтезируемый алгоритм оператор цикла `for`, который сможет обеспечить движение робота вдоль 8 сторон восьмиугольника:

Код 2.39: Синтез алгоритма движения по расходящейся спирали

```
1 from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3 while KEY.value():
4     pass
5
6 for i in range(1,8):
7     pass
8 stop()
```

После успешного запуска программы и проверки ее работоспособности без синтаксических ошибок можно добавить в тело цикла команды движения вперед и правого поворота на какой либо угол:

Код 2.40: Синтез алгоритма движения по расходящейся спирали

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

while KEY.value():
    pass

for i in range(1,9):

    LED.value(1)
    rf(100)
    st(1)
    rt(0)
    st(0.5)
stop()
```

Под управлением этой программы робот движется по траектории:

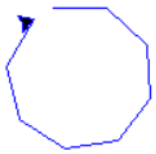


Рис. 2.58: Синтез алгоритма движения по расходящейся спирали

Далее следует подобрать время разворота таким образом, чтобы получился восьмиугольник:

Код 2.41: Синтез алгоритма движения по расходящейся спирали

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

while KEY.value():
    pass

for i in range(1,9):

    LED.value(1)
    rf(100)
    st(1)
    rt(0)
    st(0.53)
stop()
```

В результате получится траектория:

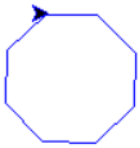


Рис. 2.59: Синтез алгоритма движения по расходящейся спирали

Анализируя заданную траекторию можно заметить, что в спирали каждая следующая сторона движения чуть больше предыдущей, это можно обеспечить применением арифметического выражения в 10 строке для последовательного увеличения времени движения вдоль очередной стороны, а значит и увеличения ее длины:

Код 2.42: Синтез алгоритма движения по расходящейся спирали

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  while KEY.value():
4      pass
5
6  for i in range(1,9):
7
8      LED.value(1)
9      rf(100)
10     st(1+0.02*i)
11     rt(0)
12     st(0.53)
13 stop()
```

Под управлением этого алгоритма робот будет двигаться по траектории:

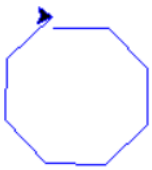


Рис. 2.60: Синтез алгоритма движения по расходящейся спирали

Для формирования спирали необходимо добавить количество повторений тела цикла:

Код 2.43: Синтез алгоритма движения по расходящейся спирали

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

while KEY.value():
    pass

for i in range(1,50):

    LED.value(1)
    rf(100)
    st(1+0.02*i)
    rt(0)
    st(0.53)
stop()
```

При этом робот будет двигаться по траектории:

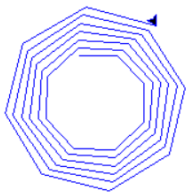


Рис. 2.61: Синтез алгоритма движения по расходящейся спирали

Для выравнивания спирали следует уточнить время разворота робота:

Код 2.44: Синтез алгоритма движения по расходящейся спирали

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED

while KEY.value():
    pass

for i in range(1,50):

    LED.value(1)
    rf(100)
    st(1+0.02*i)
    rt(0)
    st(0.5265)
stop()
```

Это приведет к следующей траектории движения робота:

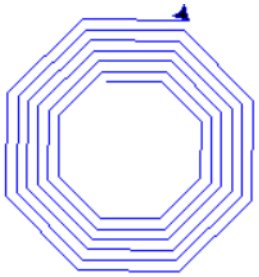


Рис. 2.62: Синтез алгоритма движения по расходящейся спирали

На основе синтезированного кода можно более подробно рассмотреть использование логических выражений. Делать это удобно на задаче управления светодиодом. Пусть необходимо, чтобы светодиод зажигался только на третьем отрезке движения робота, сделать это можно с помощью такого логического выражения (строка 8):

Код 2.45: Использование логических выражений

```
1  from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3
4  while KEY.value():
5      pass
6
7  for i in range(1,50):
8      if i==3:
9          LED.value(0)
10     else:
11         LED.value(1)
12     rf(100)
13     st(1+0.02*i)
14     rt(0)
15     st(0.5265)
16
17 stop()
```

Результат:

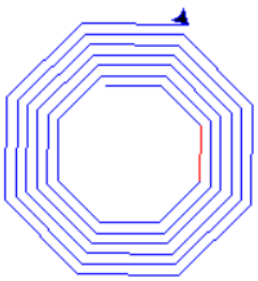


Рис. 2.63: Использование логических выражений

Если наоборот надо чтобы светодиод не горел только на третьем отрезке движения робота, сделать это можно так:

Код 2.46: Использование логических выражений

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass

for i in range(1,50):
    if i!=3:
        LED.value(0)
    else:
        LED.value(1)
    rf(100)
    st(1+0.02*i)
    rt(0)
    st(0.5265)

stop()
```

Или так:

Код 2.47: Использование логических выражений

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass

for i in range(1,50):
    if not i==3:
        LED.value(0)
    else:
        LED.value(1)
    rf(100)
    st(1+0.02*i)
    rt(0)
    st(0.5265)

stop()
```

Результат один и тот же:

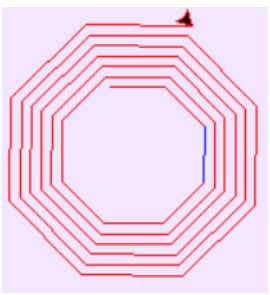


Рис. 2.64: Использование логических выражений

Добиться того, чтобы светодиод горел на четных отрезках из первых десяти можно так:

Код 2.48: Использование логических выражений

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass

for i in range(1,50):
    if i%2==0 and i<=10:
        LED.value(0)
    else:
        LED.value(1)
    rf(100)
    st(1+0.02*i)
    rt(0)
    st(0.5265)

stop()
```

Результат:

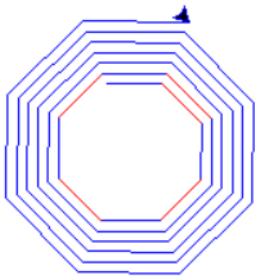


Рис. 2.65: Использование логических выражений

И последний пример, светодиод горит на нечетных отрезках или на отрезках с номером больше 10:

Код 2.49: Использование логических выражений

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass

for i in range(1,50):
    if i%2==1 or i>10:
        LED.value(0)
    else:
        LED.value(1)
    rf(100)
    st(1+0.02*i)
    rt(0)
```

```
15      st(0.5265)
16
17  stop()
```

Сама траектория:

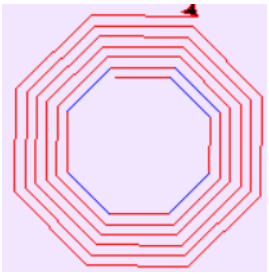


Рис. 2.66: Использование логических выражений

Более подробно о логических выражениях можно прочитать в каком-либо учебнике по языку Python.

Задания

1. Приведите примеры использования логических операций в реальной жизни.
2. Запрограммируйте робота используя код 2.37. Подберите значения переменных `r` и `k`, чтобы получалась ровная восьмерка и робот останавливался в той же точке, что и стартовал.
3. Запрограммируйте робота, чтобы он проехал по траектории, изображенной на рисунке 2.67 с учетом принятых ранее обозначений. Нарисуйте блок-схему.

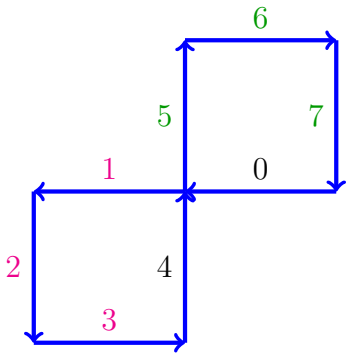


Рис. 2.67: Схема движения

4. Запрограммируйте робота, чтобы он проехал по траектории, изображенной на рисунке 2.68 с учетом принятых ранее обозначений. Нарисуйте блок-схему.

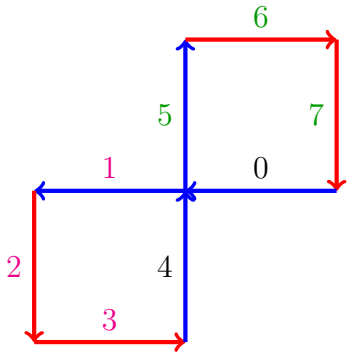


Рис. 2.68: Схема движения

5. Запрограммируйте робота, чтобы он проехал по траектории, изображенной на рисунке 1.72 с применением изученных алгоритмических конструкций. Нарисуйте блок-схему.

6. Запрограммируйте робота, чтобы он проехал по схеме:

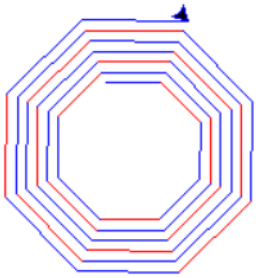


Рис. 2.69: Схема движения

Нарисуйте блок-схему.

2.9 Цикл, вложенный в цикл

Пусть необходимо, чтобы робот проехал по квадрату, изображенному на рисунке 2.70.

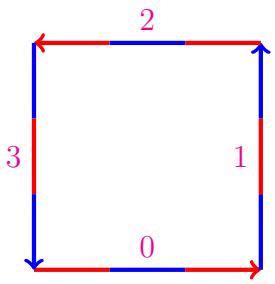


Рис. 2.70: Схема движения

При движении по сторонам квадрата робот мигает светодиодом. Реализовать такую схему движения можно использовав цикл внутри цикла. С помощью внешнего цикла робот движется по квадрату. При помощи внутреннего цикла робот мигает светодиодом (код 2.50).

Код 2.50: Цикл в цикле

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
k=1.05
i=0
while i<3:
    rf(100)
    j=0
    while j<3:
        LED.value(1-LED.value())
        st(1)
        j=j+1
    lt(0)
    st(k)
    i=i+1
stop()
```

Соответствующая блок-схема изображена на рисунке 2.71

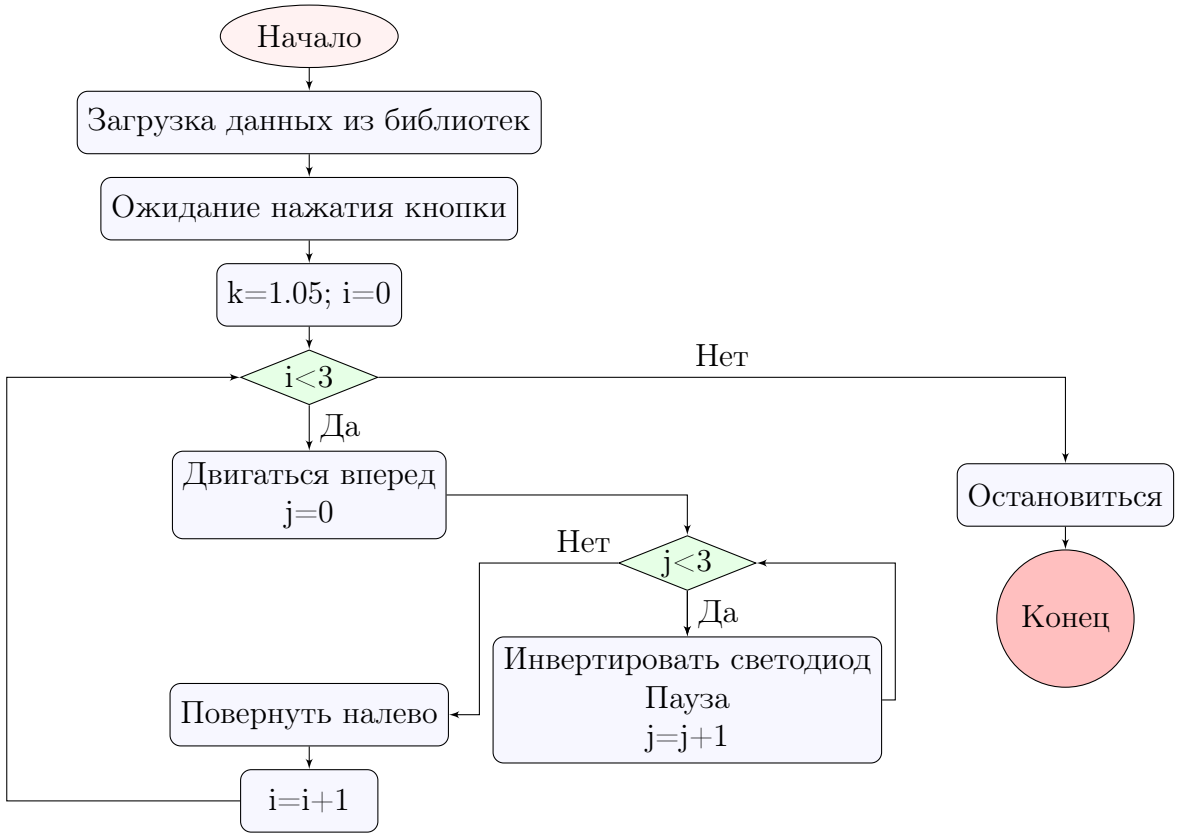


Рис. 2.71: Цикл в цикле

Задания

1. Приведите примеры использования циклов в циклах в реальной жизни.

2. Запрограммируйте робота используя код 2.50. Подберите значение переменной `k`, чтобы получался ровный квадрат и робот останавливался в той же точке, что и стартовал. Объясните поведение робота.

3. Запрограммируйте робота, чтобы он проехал по схеме, изображенной на рисунке 2.72. Составьте блок-схему алгоритма.

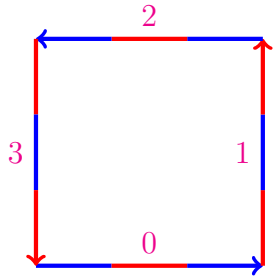


Рис. 2.72: Схема движения

4. Запрограммируйте робота, чтобы он проехал по схеме:

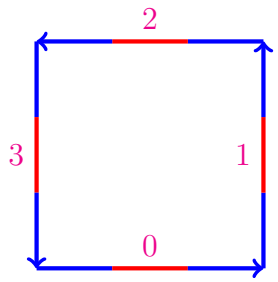


Рис. 2.73: Схема движения

Составьте блок-схему алгоритма.

2.10 Цикл, вложенный в ветвление

Цикл также может использоваться внутри ветвления. Например, необходимо, чтобы робот проехал по схеме, изображенной на рисунке 2.74.

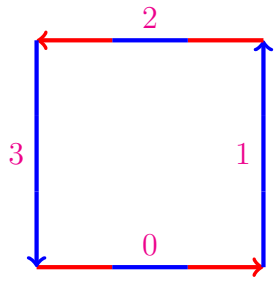


Рис. 2.74: Схема движения

Этого можно достичь, если внутри цикла, обеспечивающего движение робота по квадрату, поместить условный оператор, а внутри него - цикл, обеспечивающий мигание светодиода:

Код 2.51: Цикл, вложенный в ветвление

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
k=1.05
i=0
while i<3:
    LED.value(1)
    rf(100)
    j=0
    if(i%2)==0:
        while j<3:
            LED.value(1-LED.value())
            st(1)
```

```
16                j=j+1
17      else:
18           st(3)
19      lt(0)
20      st(k)
21      i=i+1
22  stop()
```

Светодиод будет мигать только при движении вдоль тех сторон квадрата, для которых остаток от деления значения счетчика цикла на два равен нулю. Блок схема алгоритма приведена на рисунке 2.75.

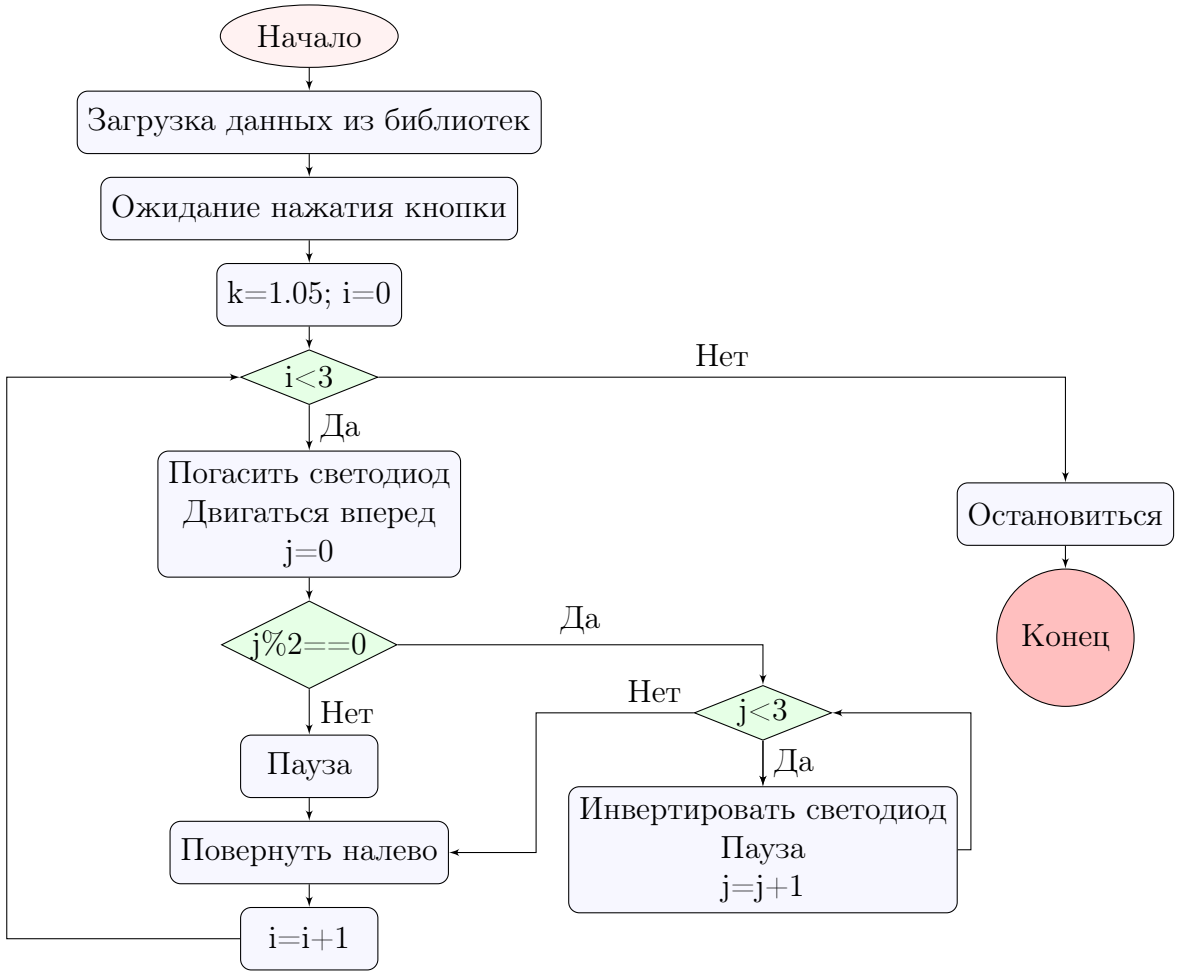


Рис. 2.75: Цикл, вложенный в ветвление

Задания

1. Приведите примеры использования циклов внутри ветвлений в реальной жизни.

2. Запрограммируйте робота используя код 2.51. Подберите значение переменной `k`, чтобы получился ровный квадрат. Объясните поведение робота.

3. Запрограммируйте робота, чтобы он проехал по схеме, изображенной на рисунке 2.76. Составьте блок-схему алгоритма.

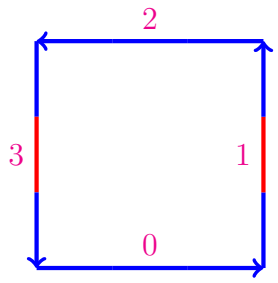


Рис. 2.76: Схема движения

4. Запрограммируйте робота, чтобы он проехал по схеме:

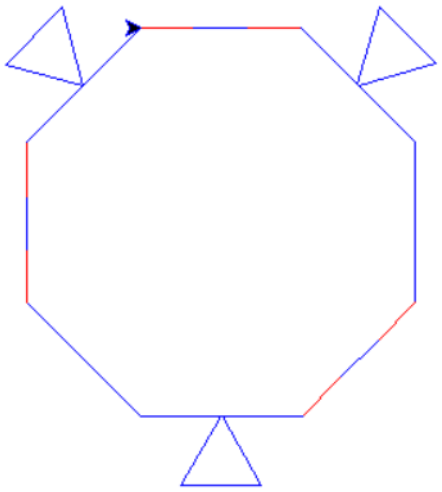


Рис. 2.77: Схема движения

Нарисуйте блок-схему.

2.11 Вспомогательные алгоритмы, функции

Выше приведены основные алгоритмические конструкции, позволяющие сделать код короче и проще. Наряду с этими конструкциями существует еще один способ сокращения кода - выделения повторяющихся фрагментов кода, которые невозможно упаковать в цикл, во вспомогательные алгоритмы.

Пусть необходимо, чтобы робот двигался по схеме:

Код, который реализует такое поведение:

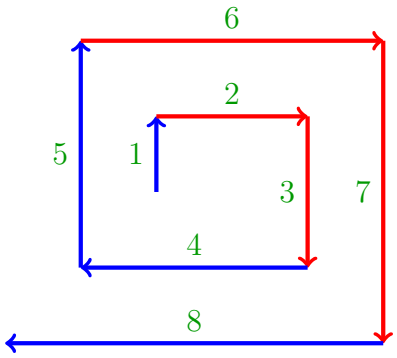


Рис. 2.78: Схема движения

Код 2.52: Код, реализующий спираль

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while KEY.value():
    pass
r=1.05
rf(100)
st(1)
lt(0)
st(r)
LED.value(1-LED.value())
rf(100)
st(2)
lt(0)
st(r)
rf(100)
st(2)
lt(0)
st(r)
LED.value(1-LED.value())
rf(100)
st(3)
lt(0)
st(r)
rf(100)
st(3)
lt(0)
st(r)
LED.value(1-LED.value())
rf(100)
st(4)
lt(0)
st(r)
rf(100)
st(4)
lt(0)
st(r)
LED.value(1-LED.value())
```

```
rf(100)
st(5)
lt(0)
st(r)
stop()
```

Внимательно изучив этот код, можно обнаружить, что в нем много раз повторяется последовательность команд:

```
lt(0)
st(l)
```

но в цикл ее оформить не очень просто. В этом случае можно воспользоваться оформлением вспомогательного алгоритма. В языке Python эти алгоритмы называются функциями. В этом случае код 2.52 может быть переписан в виде:

Код 2.53: Использование функций

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


def lft():
    lt(0)
    st(1.05)

while KEY.value():
    pass
rf(100)
st(1)
lft()
LED.value(1-LED.value())
rf(100)
st(2)
lft()
rf(100)
st(2)
lft()
LED.value(1-LED.value())
rf(100)
st(3)
lft()
rf(100)
st(3)
lft()
LED.value(1-LED.value())
rf(100)
st(4)
lft()
rf(100)
st(4)
lft()
LED.value(1-LED.value())
rf(100)
st(5)
lft()
```

```
38 stop()
```

Видно, что код стал короче.

Функция в Python – это блок кода, который начинается с ключевого слова `def`, названия функции и двоеточия, пример:

```
def lft():
    lt(0)
    st(1.05)
```

В основном коде вызов функции осуществляется через ее имя - строки 12, 16, 19 и т.д. кода 2.53

Задания

1. Изучите, пользуясь каким-либо учебником по Python, как происходит передача параметров в функции и как происходит возвращение значения функций.

2. Запрограммируйте робота используя код 2.53. Добейтесь, чтобы робот делал повороты под прямым углом. Нарисуйте блок-схему.

3. Запрограммируйте робота, чтобы он проехал по траектории, согласно схеме:

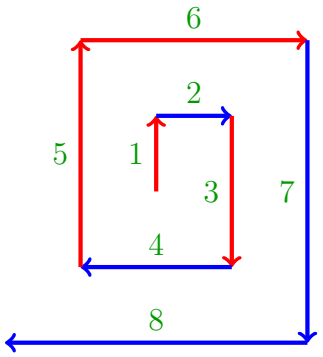


Рис. 2.79: Схема движения

Нарисуйте блок-схему.

4. Запрограммируйте робота, чтобы он проехал по траектории, согласно схеме:

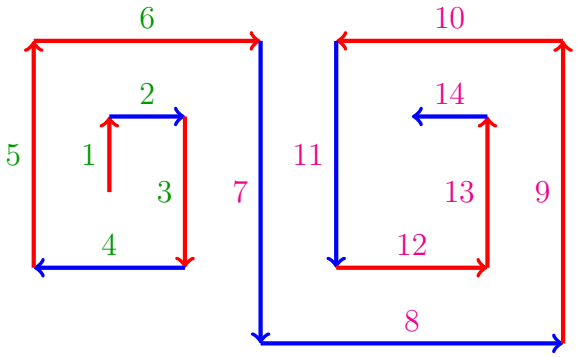


Рис. 2.80: Схема движения

2.12 Табличные величины, массивы

Помимо вышеприведенных алгоритмических конструкций уменьшению программного кода может способствовать правильная организация данных. Одними из алгоритмических объектов, которые могут помочь в этом деле являются табличные величины, которые на языке Python реализуются, например, с помощью массивов. С использованием массивов код 2.53 может быть переписан в следующем виде:

Код 2.54: Использование массивов

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED
import array

def lft():
    lt(0)
    st(1.05)

arr=array.array('i',[0,1,0,1,0,1,0,1,1,2,2,3,3,4,4,5])

while KEY.value():
    pass

for i in range(0,8):
    if arr[i]:
        LED.value(1-LED.value())
    rf(100)
    st(arr[8+i])
    lft()
stop()
```

Как видно, применение правильной организации данных (использования массивов) позволило сократить программный код вдвое.

Необходимо отметить, что ту же самую задачу можно решить и с помощью списков:

Код 2.55: Использование массивов

```
from olly.base import st, lt, rt, rf, rr, stop, KEY, IN1, LED


def lft():
    lt(0)
    st(1.05)

arr=[[0,1,0,1,0,1,0,1],[1,2,2,3,3,4,4,5]]

while KEY.value():
    pass

for i in range(0,8):
    if arr[0][i]:
        LED.value(1-LED.value())
    rf(100)
```

```
17      st(arr[1][i])
18      lft()
19  stop()
```

Фактически, в двух последних примерах программного кода сама программа движения робота содержится в массиве данных - строке 8. А строки 13-18 являются виртуальной машиной, которые исполняют эту программу.

Важно: одну и ту же задачу можно решить различными способами; правильная организация данных способна значительно упростить алгоритм.

Задания

1. Изучите, пользуясь каким-либо учебником по Python, как работать с массивами на этом языке.
2. Сравните код 2.54 и код 2.55, что в них общего и чем они различаются.
3. Запрограммируйте робота используя код 2.54. Объясните как работает этот код. Нарисуйте блок-схему.
4. Запрограммируйте робота используя код 2.55. Объясните как работает этот код. Нарисуйте блок-схему.
5. Запрограммируйте робота, чтобы он проехал по траектории, изображенной на рисунке 2.80

Задания по главе

1. Расскажите о способах сокращения программного кода, которые вам известны.
2. Расскажите о видах циклов, которые вам известны.
3. Расскажите о видах ветвлений, которые вам известны.
4. Расскажите когда лучше применять циклы, а когда - вспомогательные алгоритмы.
5. Расскажите в чем преимущества и недостатки массивов и списков в Python.
6. Запрограммируйте робота, чтобы он проехал по схеме:

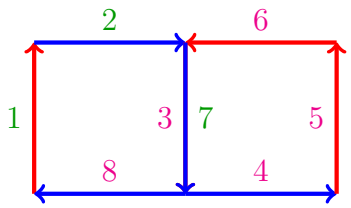


Рис. 2.81: Схема движения

Нарисуйте блок-схему.

7. Запрограммируйте робота, чтобы он проехал по схеме:

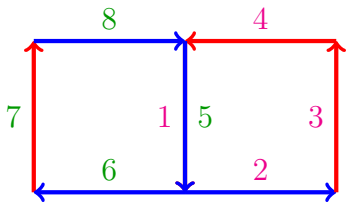


Рис. 2.82: Схема движения

Нарисуйте блок-схему.

8. Запрограммируйте робота, чтобы он проехал по схеме:

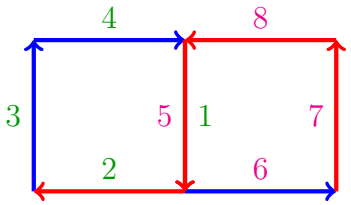


Рис. 2.83: Схема движения

Нарисуйте блок-схему.

Глава 3

Получение информации, дополнительные операторы управления

3.1 Кнопка

Все предыдущее рассмотрение касалось ситуаций когда робот почти не реагировал на внешние условия - просто осуществлял заранее заданную схему движения. В реальной жизни пользы от таких роботов не много. Важно, чтобы робот мог реагировать на какие то внешние условия, для этого он должен уметь получать информацию от датчиков.

Самым простым примером датчика является кнопка. Робот должен уметь получать информацию о том нажата кнопка или нет. Фактически в предыдущих примерах использовалось получение информации от кнопки, хотя внимание на этом не акцентировалось.

Речь идет о фрагменте кода:

```
while KEY.value():
    pass
```

Он представляет из себя цикл в котором проверяется условие

`KEY.value()`.

Если кнопка не нажата это условие равно единице, которое воспринимается циклом как истина, и начинает выполняться тело цика, представленное оператором

`pass`.

По этому оператору робот ничего не делает, это пустой оператор, и робот возвращается опять к проверке условия

`KEY.value()`.

Так продолжается до тех пор, пока на роботе не будет нажата кнопка. Как только кнопка будет нажата, условие будет равно нулю и цикл закончится, робот перейдет к выполнению следующей команды. Ниже представлена блок-схема этого фрагмента кода:

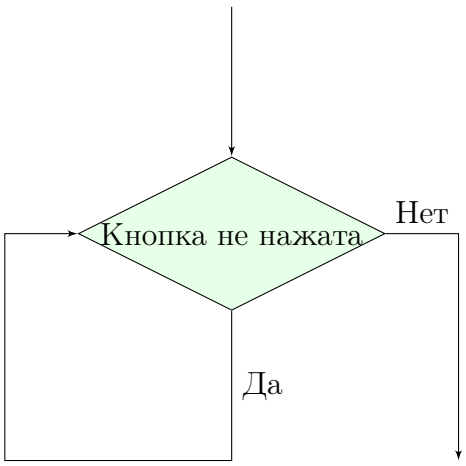


Рис. 3.1: Цикл ожидания нажатия кнопки

Следующий код демонстрирует один из способов использования информации о статусе кнопки (нажата или нет). Следует обратить внимание, что в случае использования симулятора будет использоваться второй из встроенных в Olly, поскольку импорт команд управления роботом происходит из модуля nbase.

Код 3.1: Продвинутый пример использования кнопки

```
1  from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2
3  g=1.04
4  while 1:
5      rf(0)
6      while KEY.value():
7          pass
8      for i in range(1,4):
9          LED.value(1-LED.value())
10         rf(100)
11         st(1.5)
12         rt(0)
13         st(g)
```

Задания

1. Нарисуйте блок-схему алгоритма, представленного в коде 3.1.
2. Объясните использование команды `stop()` в этом коде.
3. Попробуйте предсказать поведение робота, работающего под управлением этого кода.
4. Запрограммируйте робота, используя этот код. Объясните поведение робота.

Оператор печати

Для отладки программ необходимо уметь получать информацию о внутреннем состоянии робота во время его работы. В этом может помочь оператор печати, реали-

зованный с помощью функции `print()`. Например, следующий код выводит каждую секунду информацию о том нажата ли кнопка робота или нет:

Код 3.2: Печать информации о кнопке

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while 1:
    if KEY.value():
        print("кнопка не нажата")
    else:
        print("кнопка нажата")
    st(1)
```

В этом примере оператор выводит строковые значения. Можно модифицировать код таким образом, чтобы он выводил цифровые значения:

Код 3.3: Печать информации о кнопке

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while 1:
    print(KEY.value())
    st(1)
```

Кроме того оператор печати умеет выводить несколько значений одновременно:

Код 3.4: Печать информации о кнопке

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while 1:
    print("Состояние кнопки",KEY.value())
    st(1)
```

Следует отметить, что оператор печати работает только когда робот подключен к компьютеру, а программа запущена в терминальном режиме и вывод осуществляется на экран компьютерного монитора. Если же робот работает в автономном режиме следует для индикации режима работы робота использовать встроенный светодиод:

Код 3.5: Индикация нажатия кнопки с помощью светодиода

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while 1:
    if KEY.value():
        LED.value(1)
    else:
        LED.value(0)
    st(1)
```

Или более короткий вариант:

Код 3.6: Индикация нажатия кнопки с помощью светодиода

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while 1:
    LED.value(KEY.value())
    st(1)
```

Задания

1. С помощью какого-либо учебника по Python изучите применение оператора печати.
2. Для кодов 3.2-3.6 нарисуйте блок-схемы алгоритмов.
3. Предскажите поведение робота, запрограммированного каждым из этих кодов.
4. Запрограммируйте робота с использованием каждого кода 3.2-3.6.

3.2 Датчик отраженного света

Датчик отраженного света служит для определения цвета поверхности на которой находится робот. Информацию от этого датчика можно получить с помощью команды `IN1.read()`. Пример ее использования приведен в коде:

Код 3.7: Использование датчика отраженного света

```
from olly.nbase import IN1
import time
while True:
    print(IN1.read())
    time.sleep(1)
```

Чем светлее поверхность тем большее значение будет печатать программа, чем темнее поверхность тем меньшее значение программа будет выводить на экран компьютерного монитора.

Если робот работает в автономном режиме и команда печати недоступна, значение интенсивности отраженного света можно сохранить в переменной, а затем использовать при необходимости, например, для управления светодиодом:

Код 3.8: Использование датчика отраженного света для управления светодиодом

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED


while 1:
    intn=IN1.read()
    if intn>2000:
        LED.value(0)
    else:
        LED.value(1)
    st(1)
```

Задания

1. Для приведенных алгоритмов нарисуйте блок-схемы.
2. Запрограммируйте робота с использованием кода 3.7. Исследуйте показания датчика при расположении робота на:
   - белой поверхности;
   - черной поверхности;
   - серой поверхности;
   - поверхностях различных цветов;
   - границе черного и белого.
3. Запрограммируйте робота с использованием кода 3.8. Исследуйте поведение робота.

3.3 Ультразвуковой датчик расстояния

С помощью ультразвукового датчика расстояния робот может определять расстояние до препятствий. Функция измерения расстояния `dstnc()` находится в модуле `bus`. Пример кода измеряющего расстояние и выводящего значение измеренного расстояния представлен ниже:

Код 3.9: Использование ультразвукового датчика

```
from time import sleep
from olly.bus import dstnc

while True:

    print('Distance:', dstnc(), 'cm')
    sleep(1)
```

Результат измерения расстояния робот получает в сантиметрах. Если работ работает в автономном режиме, то измеренное расстояние можно использовать для управления светодиодом:

Код 3.10: Использование датчика расстояния для управления светодиодом для управления светодиодом

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
from olly.bus import dstnc

while True:
    l=dstnc()
    if l>15:
        LED.value(1)
    else:
        LED.value(0)
    st(1)
```

Задания

1. Для приведенных алгоритмов нарисуйте блок-схемы.

2. Объясните чем вызвано различное использование функции задержки в этих двух кодах.

3. Запрограммируйте робота с использованием кода 3.9. Исследуйте показания датчика при:

   - расстоянии до препятствия 20 см;
   - измерении расстояния до потолка;
   - расстоянии до препятствия 0 см;

4. Определите минимальное измеряемое расстояние.

5. Определите максимальное измеряемое расстояние.

6. Запрограммируйте робота с использованием кода 3.9. Исследуйте работу датчика в автономном режиме.

3.4 Гироскоп

Самый сложный датчик, входящий в состав робота - MPU6050, фактически представляющий из себя комбинацию датчика линейных ускорений, датчика угловых скоростей и датчика температуры. В дальнейшем для простоты этот датчик будет называться гороскопом. Пример получения данных от гироскопа приведен в коде:

Код 3.11: Использование ультразвукового датчика

```
from time import sleep
from olly.bimu import imu
while True:
    ax=round(imu.accel.x,2)
    ay=round(imu.accel.y,2)
    az=round(imu.accel.z,2)
    gx=round(imu.gyro.x)
    gy=round(imu.gyro.y)
    gz=round(imu.gyro.z)
    tem=round(imu.temperature,2)
    print(ax,"\t",ay,"\t",az,"\t",gx,"\t",gy,"\t",gz,\
          "\t",tem,end="\r")
    sleep(1)
```

Задания

1. Запрограммируйте робота с использованием кода 3.11. Исследуйте показания датчика при различных положениях робота.

3.5 Оператор try

Датчики робота представляют из себя отдельные от микроконтроллера электронные модули, связанные с микроконтроллером цифровыми каналами связи, работающими на высокой скорости. При передаче информации от датчика к микроконтроллеру могут возникать ошибки, при этом программа, записанная в память робота аварийно прекращает свою работу. Если такое происходит можно использовать специальный оператор `try`:

Код 3.12: Использование оператора try

```
1  from time import sleep
2  from olly.bimu import imu
3  while True:
4      try:
5          ax=round(imu.accel.x,2)
6          ay=round(imu.accel.y,2)
7          az=round(imu.accel.z,2)
8          gx=round(imu.gyro.x)
9          gy=round(imu.gyro.y)
10         gz=round(imu.gyro.z)
11         tem=round(imu.temperature,2)
12     except Exception:
13         pass
14     else:
15         print(ax,"\t",ay,"\t",az,"\t",gx,"\t",gy,"\t",gz,\
16               "\t",tem,"\t",end="\r")
17         sleep(1)
```

В общем случае его структура такова:

```
try:
    блок операторов 1
except:
    блок операторов 2
else:
    блок операторов 3
finally:
    блок операторов 4
```

Вначале программа пытается выполнить `блок операторов 1`, если при этом возникает ошибка, то выполняется `блок операторов 2`. Если при выполнении `блок операторов 1` ошибки не возникает, то далее выполняется `блок операторов 3`. И, наконец, в любом случае выполняется `блок операторов 4`.

В коде 3.12 в начале тела бесконечного цикла осуществляется попытка получить данные от гироскопа (строки 5-11). Если при этом возникает ошибка, то выполнение тела цикла прекращается и оно начинает выполняться с начала. Если при чтении данных гироскопа ошибок не возникает, то данные от гироскопа выводятся на экран монитора, программа делает секундную паузу и переходит к новому выполнению тела цикла.

Задания

1. С помощью какого-либо учебника по Python изучите применение оператора try.

2. Нарисуйте блок-схему алгоритма, представленного кодом 3.12.

3.6 Использование `break`, `else`, `continue` в циклах

Код 3.12 может быть перписан в более коротком виде:

Код 3.13: Использование оператора continue

```
from time import sleep
from olly.bimu import imu
while True:
    try:
        ax=round(imu.accel.x,2)
        ay=round(imu.accel.y,2)
        az=round(imu.accel.z,2)
        gx=round(imu.gyro.x)
        gy=round(imu.gyro.y)
        gz=round(imu.gyro.z)
        tem=round(imu.temperature,2)
    except Exception:
        continue
    print(ax,"\t",ay,"\t",az,"\t",gx,"\t",gy,"\t",gz,\
              "\t",tem,"\t",end="\r")
    sleep(1)
```

Оператор `continue` начинает следующий проход цикла, минуя оставшееся тело цикла (`while` или `for`).

Если после ошибки чтения показаний датчика нет надежды восстановить его работоспособность необходимо прекратить повторные попытки чтения показаний датчика - выйти из цикла. Для этого можно воспользоваться оператором `break`, который прерывает выполнение цикла:

Код 3.14: Использование оператора continue

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
from time import sleep
from olly.bimu import imu
while True:
    try:
        ax=round(imu.accel.x,2)
        ay=round(imu.accel.y,2)
        az=round(imu.accel.z,2)
        gx=round(imu.gyro.x)
        gy=round(imu.gyro.y)
        gz=round(imu.gyro.z)
        tem=round(imu.temperature,2)
    except Exception:
        break
    print(ax,"\t",ay,"\t",az,"\t",gx,"\t",gy,"\t",gz,\
              "\t",tem,"\t",end="\r")
    sleep(1)
while 1:
    LED.value(1-LED.value())
    sleep(0.3)
```

Задания

1. С помощью какого-либо учебника по Python изучите применение `break, else, continue` в циклах.

2. Нарисуйте блок-схемы алгоритмов, представленных кодами 3.12-3.13.

3. Самостоятельно составьте программу с использованием раздела `else` в цикле.

3.7 Потоки

Все предыдущее рассмотрение касалось случаев, где все команды выполнялись в одном вычислительном потоке. Иногда бывает удобно, чтобы различные вычислительные процессы выполнялись независимо друг от друга. Пример такого алгоритма представлен ниже:

Код 3.15: Использование потоков

```
1  from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2  from time import sleep
3  from olly.bus import dstnc
4  import _thread
5
6  def testThread():
7    while True:
8      LED.value(1-LED.value())
9      sleep(0.3)
10
11 _thread.start_new_thread(testThread, ())
12
13 while True:
14
15     print('Distance:', dstnc(), 'cm')
16     sleep(1)
```

В этом примере выполняется два потока. В основном с одной периодичностью выводятся данные от датчика расстояния. Во втором потоке с другой периодичностью осуществляется мигание светодиодом. Оба потока выполняются независимо друг от друга - асинхронно.

Организация потока осуществляется с помощью импортирования модуля `_thread` (строка 4); определения функции с описанием потока (строки 6-9); запуска потока (строка 11).

Задания

1. С помощью какого-либо учебника по Python изучите применение потоков.

2. Нарисуйте блок-схему алгоритма, представленного в коде 3.15.

3. Запрограммируйте робота с использованием кода 3.15.

Глава 4

Автономные роботы

4.1 Движение вдоль линии

Одно из заданий для автономного робота - движение вдоль черной линии, нарисованной на белом фоне. При этом робот ставится на линию так, чтобы датчик отраженного света располагался на левой границе черной линии. Робот должен быть запрограммирован так, чтобы он двигался вдоль этой границы.

При этом если робот отклоняется влево - т.е. датчик оказывается над белым фоном и его показания растут необходимо, чтобы он начал разворачиваться вправо. Если робот отклоняется вправо - т.е. датчик оказывается над черным фоном и и его показания падают необходимо, чтобы он начал разворачиваться влево.

Для написания программы необходимо определить чему равно показание датчика когда он располагается на границе черного и белого. Для этого можно воспользоваться кодом 3.7, расположив датчик над границей и сняв показания датчика.

Пусть это значение равно 1950. Тогда алгоритм движения по черной линии может выглядеть следующим образом:

Код 4.1: Движение по черной линии

```
from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
#from bus import dstnc
import _thread

def LedThread():
    while True:
        LED.value(1-LED.value())
        st(0.3)

while KEY.value():
    pass

_thread.start_new_thread(LedThread, ())

rf(100)
while True:
    if IN1.read()>1950:
        rt(1)
    else:
        lt(1)
    #print('Distance:', dstnc(), 'cm')
```

```
22      st(0.2)
```

Задания

1. Нарисуйте блок-схему алгоритма, представленного в коде 4.1.
2. Запрограммируйте робота, используя код 4.1. Объясните как он работает.
3. Попробуйте улучшить алгоритм, увеличив скорость движения робота.

4.2 Движение по черной линии с объездом препятствий

Если на линии расположить препятствие, например, банку из-под лимонада, то робот должен уметь объехать это препятствие. Для этого необходимо задействовать датчик расстояния. Ниже представлен вариант кода, реализующего такое поведение робота:

Код 4.2: Движение по черной линии с объездом препятствий

```
1  from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2  from olly.bus import dstnc
3  import _thread
4
5  d=60
6
7  def DThread():
8      global d
9      while True:
10         d=dstnc()
11         st(0.1)
12
13 def mn():
14     lt(1)
15     st(1)
16     rt(3)
17     st(4)
18
19 while KEY.value():
20     pass
21
22 _thread.start_new_thread(DThread, ())
23
24 rf(100)
25 while True:
26     if (d>0)and(d<15):
27         mn()
28     if IN1.read()>1950:
29         rt(1)
30     else:
31         lt(1)
32     st(0.2)
```

Здесь расстояние измеряется в отдельном потоке и записывается в переменную d (строки 8-11). В основном цикле (строки 24-32) проверяется нет ли в непосредственной близости препятствия и если есть, то выполняется маневр объезда (строки 13-17). Маневр объезда необходимо настроить подобрав значения времен задержек (строки 15 и 17) и радиусов поворота (строки 14 и 16).

Для более качественного движения по линии строки 28-32 так же следует настроить.

Задания

1. Нарисуйте блок-схему алгоритма, представленного в коде 4.2.
2. Запрограммируйте робота, используя код 4.2. Объясните как он работает.
3. Запрограммируйте робота, чтобы одно препятствие он объезжал слева, а другое - справа.

4.3 Сумо

В соревнованиях по сумо робот должен двигаясь внутри круга найти робота противника и вытолкнуть его из круга. Для решения этой задачи необходимо использовать два датчика - расстояния и отраженного света. Поиск можно осуществлять двигаясь по кругу. Вариант кода представлен ниже:

Код 4.3: Сумо

```
1  from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2  from olly.bus import dstnc
3  import _thread
4
5  d=60
6
7  def DThread():
8      global d
9      while True:
10         d=dstnc()
11         st(0.1)
12
13
14
15 while KEY.value():
16     pass
17
18 _thread.start_new_thread(DThread, ())
19
20 rt(0)
21 while True:
22     if (d>0)and(d<15)and(IN1.read()>1950):
23         rf(100)
24     elif (IN1.read()<=1950):
25         rr(100)
26         st(1)
27         break
```

```
28      else:
29          rt(0)
30      st(0.2)
31  stop()
```

Задания

1. Нарисуйте блок-схему алгоритма, представленного в коде 4.3.
2. Запрограммируйте робота, используя код 4.3. Объясните как он работает.
3. Попробуйте улучшить алгоритм, усовершенствовав схему поиска.

4.4 Релейный и пропорциональный регуляторы

В коде 4.1 управление движением робота осуществляется так, что робот поворачивает влево если он наехал на черную линию, либо вправо если съехал с нее. В теории управления существует понятие регулятора, под которым понимается устройство следящее за состоянием системы и реагирующее на изменение состояния с помощью каких-либо воздействий. Программа следит за состоянием с помощью датчика отраженного света, а реагирует с помощью воздействий - поворотов влево и вправо.

Сам принцип слежения за состоянием и реагированием на него называется принципом обратной связи. Этот принцип широко используется не только в технике но и в биологических системах.

В случе с роботом регулирование осуществляется таким образом, что одно из колес всегда стоит и поворот осуществляется вокруг неподвижного колеса (проанализируйте код и убедитесь в этом), поэтому робот едет вдоль черной линии со скоростью вдвое меньшей максимально возможной скорости движения робота.

Фактически с помощью кода 4.1 реализован релейный регулятор - в зависимости от состояния системы она переводится в одно из дискретных состояний - поворот налево или направо. Достоинством релейного регулятора является его простота, недостатком - значительная потеря скорости движения.

Возможен другой подход - плавно регулировать кривизну дуги поворота пропорционально отклонению от границы черной линии - построить пропорциональный регулятор. Соответствующий код представлен ниже:

Код 4.4: Пропорциональный регулятор

```
1  from olly.nbase import st, lt, rt, rf, rr, stop, KEY, IN1, LED
2  import _thread
3
4  def LedThread():
5      while True:
6          LED.value(1-LED.value())
7          st(0.3)
8
9  while KEY.value():
10     pass
11
12 _thread.start_new_thread(LedThread, ())
13 sr=1950
```

```
14 mx=2600
15 mn=900
16 rf(100)
17 while True:
18     rd=IN1.read()
19     if rd>sr:
20         rt(int((100)/(1+min(99,int(99*(rd-sr)/(mx-sr))))))
21     else:
22         lt(int((100)/(1+min(99,int(99*(rd-sr)/(mn-sr))))))
23     #print('Distance:', dstnc(), 'cm')
24     st(0.2)
```

Для достижения наилучшего результата следует настроить значения трех переменных в строках 13, 14 и 15 кода. Достоинством пропорционального регулятора является большая скорость робота, а недостатком - сложность настройки.

Задания

1. Нарисуйте блок-схему алгоритма, представленного в коде 4.4.
2. Запрограммируйте робота, используя код 4.4. Объясните как он работает.
3. Оптимизируйте алгоритм, подобрав значения переменных в строках 13, 14 и 15 кода.

4.5 Гироскоп, получение углов

В параграфе 3.4 показано как получить сырые данные от гироскопического датчика - угловые скорости и линейные ускорения. Часто бывает необходимо знать угловые положения объекта - его ориентацию, для этого следует произвести обработку данных датчика. В следующем коде показано как это можно сделать:

Код 4.5: Получение углов

```
1  import utime
2  import math
3  import umatrix
4  from olly.bimu import imu
5
6  from madgwickahrs import MadgwickAHRS
7
8
9  imu.accel_range = 0
10 imu.gyro_range = 1
11 imu.filter_range = 1
12
13 ahrs = MadgwickAHRS(sampleperiod=0.150, beta=0.05)
14
15
16 acc = []
17 gyro = []
18 while True:
```

```
19      accelerometer = imu.accel
20      gyrometer = imu.gyro
21      acc = umatrix.matrix([accelerometer.x, accelerometer.y,
        accelerometer.z], cstride=1, rstride=3, dtype=float)
22      gyro = umatrix.matrix([gyrometer.x, gyrometer.y, gyrometer.z],
        cstride=1, rstride=3, dtype=float)
23
24
25      ahrs.update_imu(gyro * math.pi/180.0, acc)
26
27      ( pitch, yaw, roll) = ahrs.quaternion.to_euler()
28
29      print("roll: %0.2f, pitch: %0.2f yaw: %0.2f\n" %(roll, pitch,
        yaw))
```

Следует отметить, что получаемые углы (в радианах) вычисляются с ошибками, увеличивающимися со временем. Для уменьшения этих ошибок необходимо провести калибровку гироскопического датчика.

Задания

1. Нарисуйте блок-схему алгоритма, представленного в коде 4.5.
2. Запрограммируйте робота, используя код 4.5. Объясните как он работает.
3. Попробуйте уменьшить получаемые ошибки.

Заключение

Основой искусства программирования является алгоритмическое мышление, умение составлять и отлаживать алгоритмы. Для их формирования необходимы регулярные занятия, которые способствуют развитию мыслительных способностей человека.

При этом не столь важен конкретный язык программирования как умение представить алгоритм с помощью блок-схемы. Хотя правильный выбор языка программирования способствует более быстрому овладению навыками написания и отлаживания программ.

Основными алгоритмическими объектами являются величины, выражения, команда присваивания, табличные величины. Основные алгоритмические конструкции - следование, ветвление, повторение. Используя эти сущности можно составить алгоритм для решения практически любой задачи.

Управление автономными объектами строится по принципу обратной связи, одному из фундаментальных принципов техники и живой природы. Он заключается в том, что с помощью датчиков получается информация о состоянии объекта, это состояние сравнивается с желаемым состоянием и на основе сравнения принимается решение о формировании управляющих воздействий на объект управления.

С помощью этой книги неленивые читатели погрузились в интересный мир управления автономными роботами, применение которых чрезвычайно велико. Дальнейшее совершенствование навыков управления возможно с помощью более глубокого освоения возможностей языка Python, изучения языков C/C++, инструментальной среды ROS - операционной среды для роботов, изучению систем машинного зрения и нейронных сетей.

Введению в ROS и использованию ее для управления роботами будет посвящено продолжение этой книги.

Список источников

[1] https://docs.zephyrproject.org/latest/boards/arm/blackpill_f411ce/doc/index.html

[2] https://github.com/lemariva/uPyIMU/tree/master

[3] https://github.com/lemariva/uPySensors/tree/aeb282aa5aa8c893fa9072b202d7a43edff4ae18

[4] https://greenteapress.com/thinkpython2/thinkpython2.pdf

[5] https://micropython.org

[6] https://stepik.org/course/58852/promo

[7] http://www.dec.su

Приложение А

Список команд робота

- rf(x) двигаться вперед со скоростью x: целое, 0<=x<=100
- rr(x) двигаться назад со скоростью x: целое, 0<=x<=100
- lt(x) разворачиваться влево, двигаясь по дуге радиусом x: целое, 0<=x<=100
- rt(x) разворачиваться вправо, двигаясь по дуге радиусом x: целое, 0<=x<=100
- st(x) сохранять текущий режим работы в течении x секунд: 0<=x
- stop() остановиться
- LED.value(0) зажечь светодиод
- LED.value(1) погасить светодиод
- LED.value() узнать не горит ли светодиод
- KEY.value() узнать не нажата ли кнопка

Приложение Б

Электрическая схема робота

Электрическая схема робота приведена на рисунке Б.1.

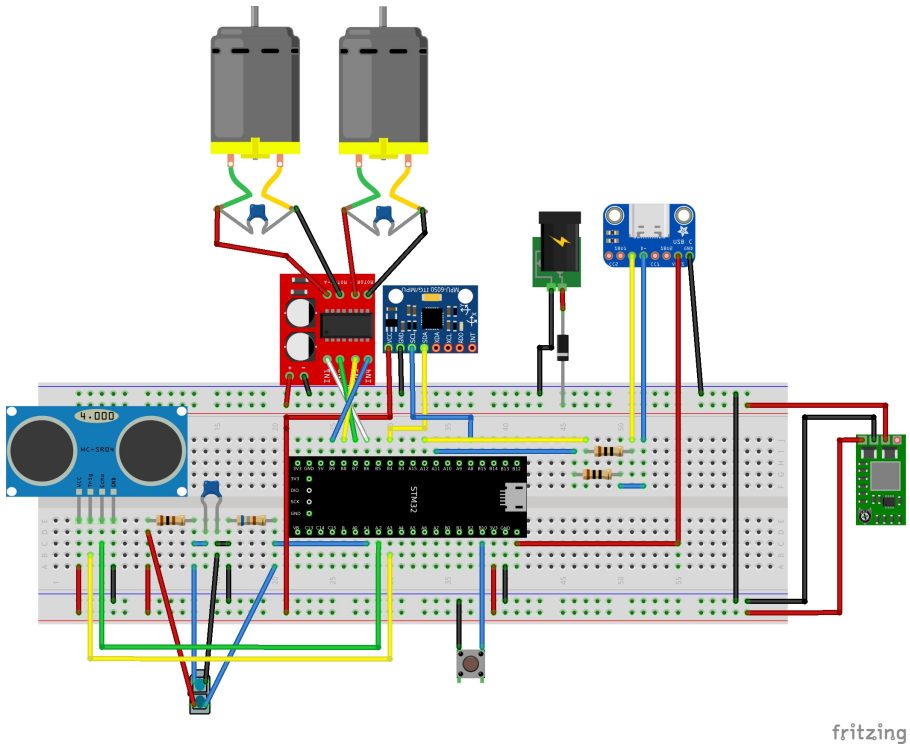


Рис. Б.1: Электрическая схема робота

В качестве программируемого вычислительного устройства выбран МК STM32F411CEU6 по причине его впечатляющих аппаратных возможностях при относительно низкой до недавнего времени цене. В своем составе МК имеет: таймеры с аппаратными квадратурными детекторами (АКД) и возможностью генерации широтно - импульсных модулированных (PWM) сигналов, аналого - цифровые преобразователи (ADC), цифровые интерфейсы универсального синхронно-асинхронного приемопередатчика (USART) и шины для связи между интегральными схемами внутри электронных приборов (I2C).

На рынке присутствуют готовые модули с этим МК, имеющие народное наименование Black Pill, которые удобно использовать при мелкомодульной сборке (ММС). Фотография этого модуля приведена на рисунке Б.2. Полезной особенностью Black Pill является наличие места под распайку внешней флеш памяти под пользовательские программы. Ресурс такой памяти составляет более 100000 циклов перезаписи, что при интенсивности использования порядка 100 циклов перезаписи в день позволит достичь срока полезной работы робота, построенного на основе выбранного МК, в течении нескольких лет. Питание МК осуществляется от источника 3,3 или 5 В.

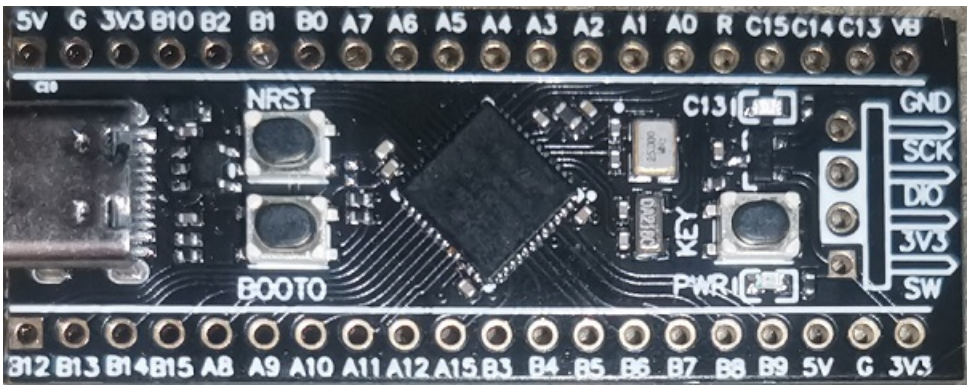


Рис. Б.2: Black Pill

В качестве датчика углового положения выбран модуль GY-521 на основе микросхемы MPU6050, представляющей из себя комбинацию 3-х осевых гироскопа и акселерометра (рисунок Б.3). Данный модуль отличается простотой и стабильностью работы и широко используется в самодельных конструкциях на основе Arduino. Для связи с МК модуль представляет хорошо зарекомендовавший себя интерфейс I2C, и имеет диапазон питания от 3,3 до 5 В.

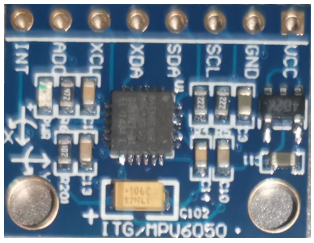


Рис. Б.3: GY-521

Датчик отраженного света построен на основе оптопары TCRT5000, состоящей из инфракрасного светодиода и фототранзистора, которая обеспечивает надежное детектирование черной линии на белом фоне при соответствующем подборе обвязывающих элементов (рисунок Б.4).

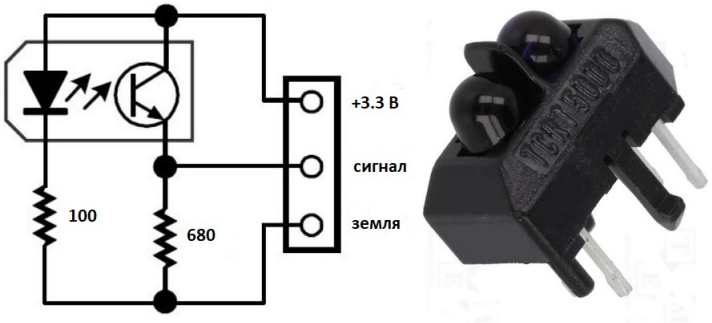


Рис. Б.4: TCRT5000

Выбранный датчик расстояния HC-SR04 широко распространен, обладает высокой точностью, стабильностью работы и имеет модификации, способные работать по USART и питаться от напряжения 3.3 В, например, модификация 2020 г. на микросхеме RCWL-9206. Диапазон измеряемых расстояний у этого варианта датчика

несколько меньше чем у 5-ти вольтовой версии – до 60 см, но этого вполне достаточно для работы школьного робота (рисунок Б.5).

Рис. Б.5: HC-SR04

Питание электронной части робота осуществляется от стабилизатора на основе микросхемы MP2315, позволяющего получать стабилизированное напряжение в широком диапазоне от 1.2 до 9 В. Фотография модуля представлена на рисунке Б.6.

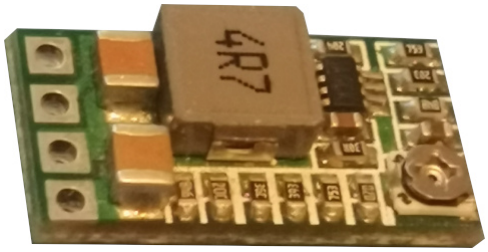


Рис. Б.6: Стабилизатор напряжения

Электромеханическую группу образуют два электродвигателя GA12-N20 3–12В, 60 об/мин. Внешний вид двигателя приведен на рисунке Б.7. Для снижения трения в редукторе, последний щедро набит водостойкой консистентной смазкой, которая обеспечивает надежную работу двигателя весь срок использования без технического обслуживания. Данный двигатель развивает большой момент для обеспечения хорошего хода робота, хотя и не на высокой, но достаточной для целей обучения программированию скорости.

При выборе драйвера двигателей предпочтение было отдано модулю на основе микросхемы MX1508 (рисунок Б.8). Модуль имеет защиту от перегрева, но, как выяснилось во время разработки, не защищен от ошибок подключения источника питания, при неправильной полярности микросхема моментально выгорает. Поэтому в электрическую схему робота включен защитный диод Шотки 1N5817.

Кроме того электрическая схема содержит два керамических конденсатора емкостью 0.1 мкф, шунтирующих электродвигатели по высокой частоте и один керамический фильтрующий конденсатор емкостью 1.0 мкф, включенный параллельно светодиоду оптопары, двухконтактный JST female разъем (рисунок Б.9), а также тактовую кнопку для управления роботом.

Совокупная стоимость перечисленных электронных компонентов на момент разработки составляла при заказе из Китая менее 1000 руб., что является хорошим результатом и позволяет создать достаточно дешевого школьного робота.

Рис. Б.7: Электродвигатель

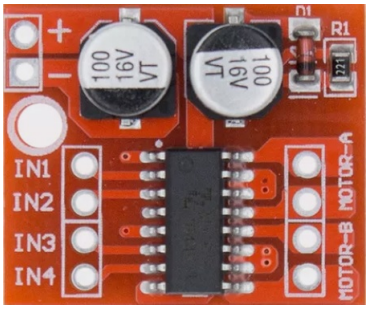


Рис. Б.8: Драйвер двигателей

Рис. Б.9: JST female разъем

Приложение B

Конструкция робота

Основные конструктивные детали робота изготовлены на 3D принтере. Технология 3D печати позволяет получать сравнительно дешевые мелкосерийные изделия, при очень высокой гибкости их модификации. Для выполнения печати разработаны соответствующие 3D модели. В качестве материала для печати выбран PETG пластик. Он обладает хорошими механическими свойствами и химически устойчив.

В результате серии доработок по результатам 10000 часов опытной эксплуатации робот приобрел следующий внешний вид (рисунок B.1).

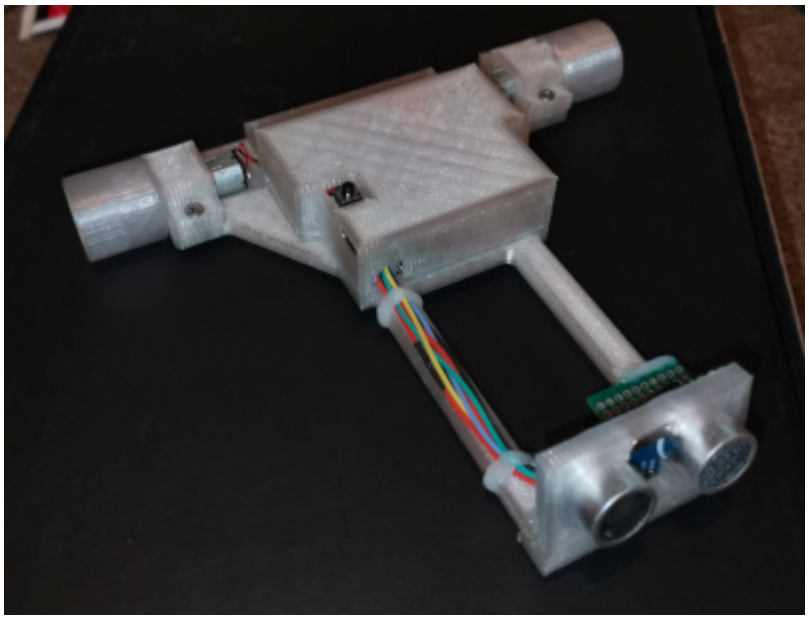


Рис. B.1: Внешний вид робота

3D модель робота находится по адресу http://www.dec.su/upload/wysiwyg/robot_model.zip[1].

Отдельные элементы робота соединяются с помощью термоклея или винтов M3 8-10мм, что позволяет легко осуществлять при необходимости их демонтаж и замену. Особенностью конструкции является то, что монтаж электромеханических элементов (кнопки, USB разъема и разъема питания) осуществляется не на верхней крышке, а на боковой стенке. Это позволяет независимо снимать верхнюю крышку, что обеспечивает простой доступ к элементам электронной схемы для их обслуживания.

Сборка робота начинается с подготовки платы стабилизатора напряжения. Вначале плата настраивается на напряжение стабилизации 3.3 вольта, для этого перере-

[1]При возникновении проблем с печатью 3d модели можно обратиться к автору для приобретения распечатанного комплекта пластиковых элементов робота.

зается перемычка ADJ и напаивается перемычка 3.3v, как показано на рисунке В.2.

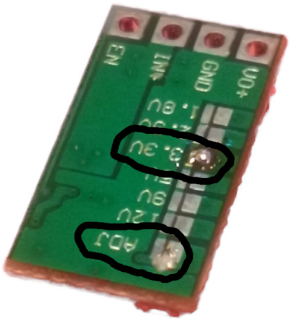


Рис. В.2: Настройка напряжения стабилизации 3.3 вольта

Затем к стабилизатору припаиваются провода питания. Вначале к левому конденсатору (рисунок В.3) припаивается пара проводов AWG22 длиной 7 см, затем к правому конденсатору - две пары проводов AWG28 длиной 4 см.

Рис. В.3: Подготовка стабилизатора напряжения (1)

Далее подготавливается разъем питания - в разрыв красного провода AWG22 впаивается диод шоттки полярностью как на рисунке В.4. Диод изолируется термоусадкой. Длина проводов питания 10 см.

Рис. В.4: Подготовка разъема питания

После чего разъем питания подпаивается к стабилизатору, кроме того к стабилизатору подпаиваются еще две пары проводов питания AWG28 длиной 18 см. (рисунок В.5).

Таким образом к стабилизатору напряжения подпаиваются: разъем питания от внешней батареи; пара проводов AWG22 для питания драйвера двигателей напряжением батареи; две пары коротких проводов AWG28 для питания микроконтроллера

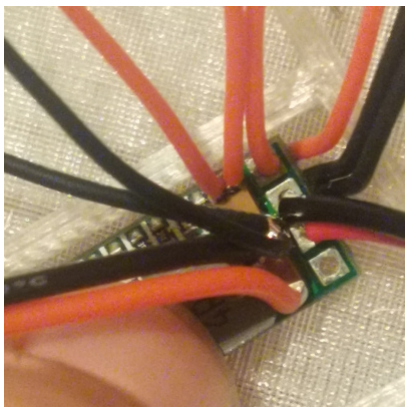


Рис. В.5: Подготовка стабилизатора напряжения (2)

и гироскопа напряжением 3.3 вольта; две пары длинных проводов AWG28 для питания датчиков робота напряжением 3.3 вольта. Подготовленный стабилизатор напряжения устанавливается на шасси робота и фиксируется с помощью термоклея (рисунок В.6).

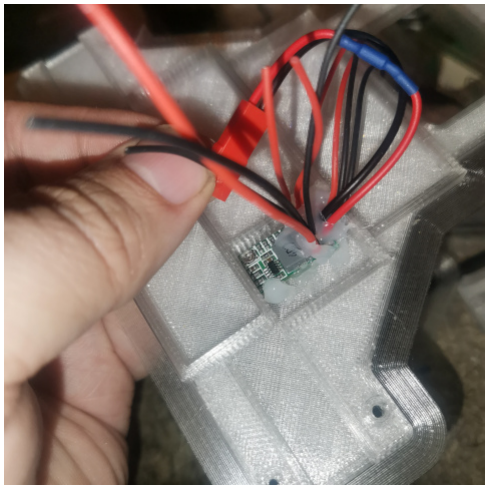


Рис. В.6: Установленный стабилизатор напряжения

Следующим на шасси вклеивается гироскоп с предварительно подпаянными сигнальными проводами AWG28 к контактам SCL (зеленый) и SDA (желтый) длиной 4 см, а также питанием 3.3 вольта - короткая пара AWG28 от стабилизатора напряжения (рисунок В.7).

Перед установкой платы микроконтроллера Black Pill на нее впаивается флеш память 25Q64FVSIG (рисунок В.8) и микроконтроллер прошивается версией MicroPython для флеш объемом 8 МБ с помощью программатора st-link. Файл прошивки можно скачать по адресу http://www.dec.su/upload/wysiwyg/firmware_robot_f411_8mb_i2c3.hex.

Питание микроконтроллера осуществляется с помощью короткой пары проводов AWG28 от стабилизатора питания (рисунок В.9).

Контакты SLC и SDA гироскопа подпаиваются к контактам A8 и B4 платы микроконтроллера соответственно.

Перед установкой электродвигателей на шасси зубья шестерен редукторов смазываются тонким слоем смазки, затем электродвигатели фиксируются на шасси с

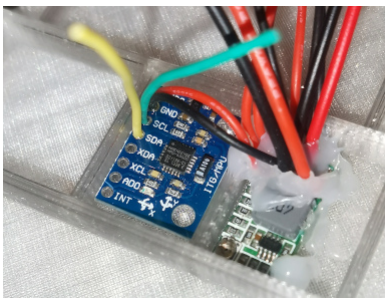


Рис. B.7: Установка гироскопа

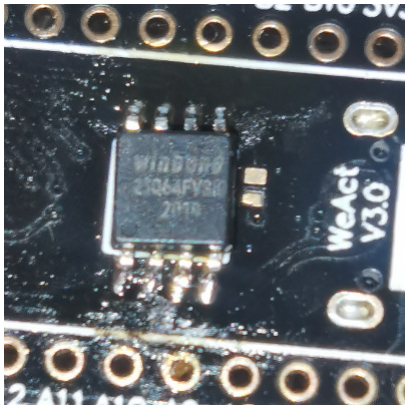


Рис. B.8: Flash память

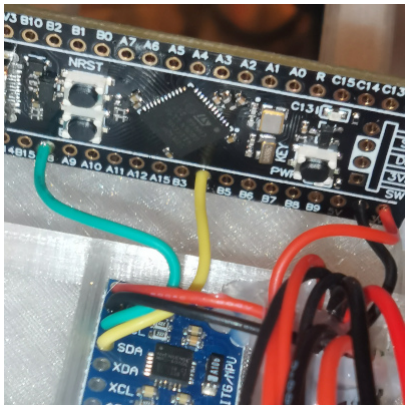


Рис. B.9: Питание микроконтроллера

помощью скоб винтами M3 10мм (рисунок B.10), и к контактам электродвигателей припаиваются керамические конденсаторы емкостью 0.1 мкф.

Выводы электродвигателей подпаиваются к драйверу согласно рисунку B.11: левый двигатель к контактам драйвера MOTOR-B, правый - MOTOR-A, соблюдая порядок подпайки красного и черного проводов от двигателей (соответствующих полюсов электродвигателей).

Питание драйвера осуществляется с помощью пары проводов AWG22 от стабилизатора напряжения напряжением внешней батареи (рисунок B.12). Управление драйвером осуществляется с контактов B6, B7, B8 и B9 платы микроконтроллера, которые подпаиваются к контактам драйвера INT1, INT2, INT3 и INT4 соответственно. После подключения драйвера он вклеивается на шасси с помощью термоклея.

Рис. В.10: Установка электродвигателя

Рис. В.11: Подключение электродвигателей к драйверу

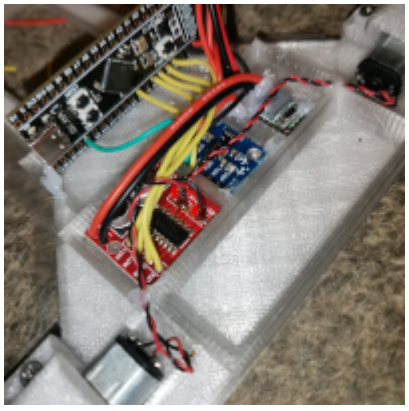


Рис. В.12: Подключение драйвера

Кроме выполненных подключений к плате микроконтроллера подпаиваются провода AWG28 к контактам: 5V - красный длиной 8 см; G - 2 черных по 8 см каждый; B10 - любой из цветов зеленый/синий/желтый 8 см; A11 - синий 8 см; A12 - желтый 8 см; A3 - желтый, A2 - зеленый, A1 - синий по 18 см каждый (рисунок В.13).

После подпайки всех проводов плата микроконтроллера фиксируется на шасси с помощью термоклея (рисунок В.14).

Далее 7 длинных проводов (2 красных, 2 черных, зеленый синий и желтый) укладываются вперед вдоль правой стороны шасси и термоклеем крепится боковая стенка

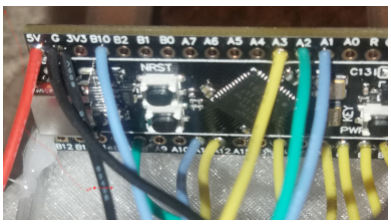


Рис. В.13: Подключение микроконтроллера

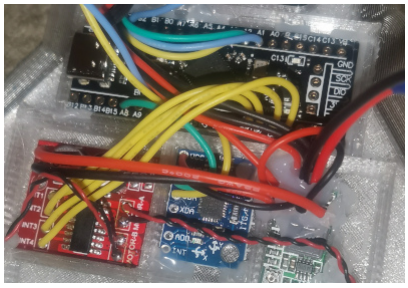


Рис. В.14: Фиксация микроконтроллера

отсека электроники (рисунок В.15).

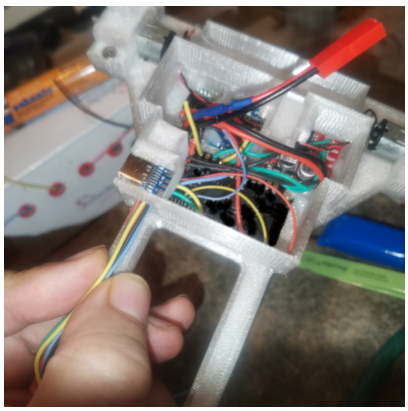


Рис. В.15: Фиксация боковой стенки отсека электроники

С помощью термоклея на специально отведенных площадках боковой стенки закрепляются кнопка и USB разъем Type-C. (рисунок В.16).

К кнопке подпаивается один короткий черный провод и один короткий синий от контактов G и B10 от платы микроконтроллера.

К разъему USB подпаиваются: короткий черный провод - к контакту G; короткий красный - от контакта 5V на плате микроконтроллера к контакту V; короткий желтый - от контакта A12 к контакту D+ через сопротивление 10 ом; короткий синий - от контакта A11 к контакту D- через сопротивление 10 ом.

Разъем питания обрезается таким образом, чтобы поместился в отведенное для него место на боковой стенке отсека электроники и вклеивается туда с помощью термоклея (рисунок В.17).

После чего крышка отсека электроники фиксируется с помощью термоклея (рисунок В.18).

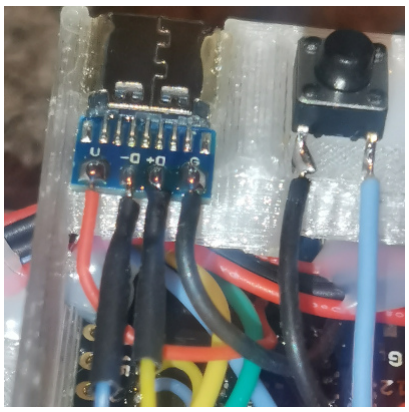


Рис. В.16: USB разъем и кнопка

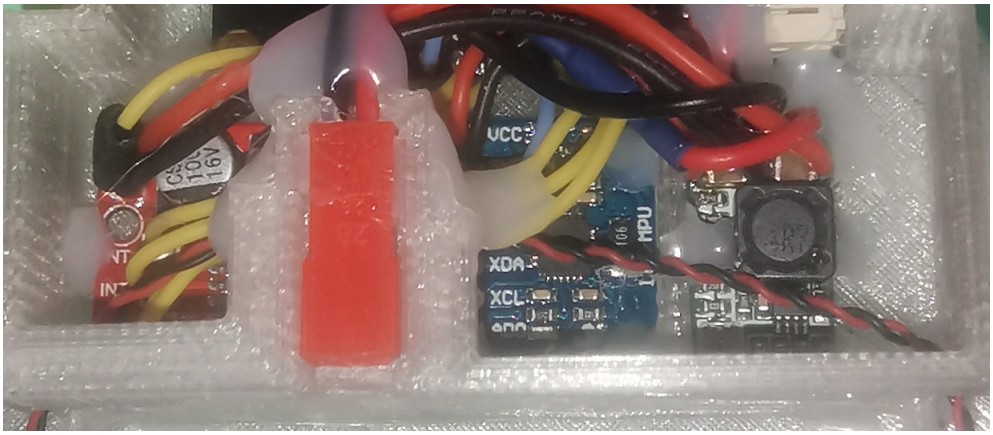


Рис. В.17: Разъем питания

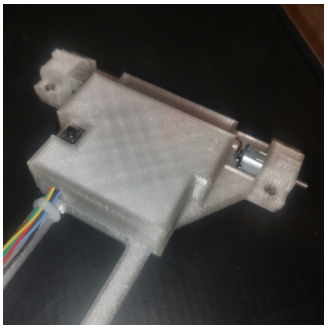


Рис. В.18: Крышка отсека электроники

Датчик отраженного света собирается на куске макетной платы по схеме, приведенной на рисунке Б.4, после чего собранный датчик вклеивается на шасси и подпаивается следующим образом: земляной контакт - черный провод, +3.3в - красный провод, сигнал - синий провод - контакт A1 на плате микроконтроллера (рисунки В.19-В.21).

В передней части шасси вместо передних колес с помощью винта М3 8мм прикручивается полусферическая опора (рисунок В.20), а также с помощью клея В-7000 фиксируется держатель ультразвукового датчика расстояния (рисунок В.22), на который вклеивается непосредственно сам датчик. К датчику подпаиваются оставши-

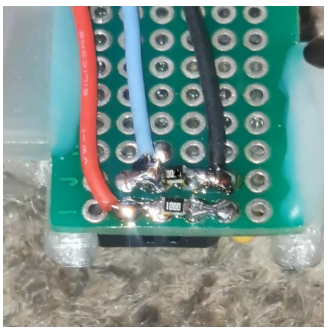


Рис. В.19: Датчик отраженного света (1)

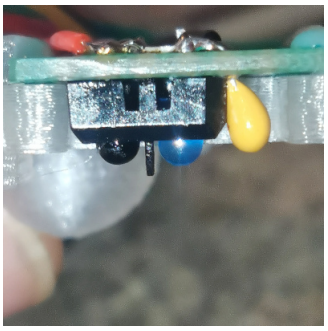


Рис. В.20: Датчик отраженного света (2)

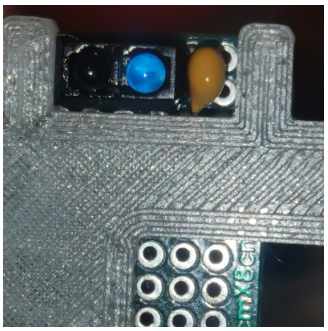


Рис. В.21: Датчик отраженного света (3)

еся четыре провода согласно рисунку В.23.

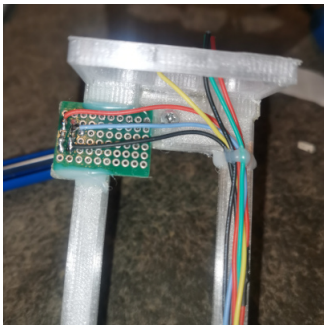


Рис. В.22: Держатель датчика расстояния

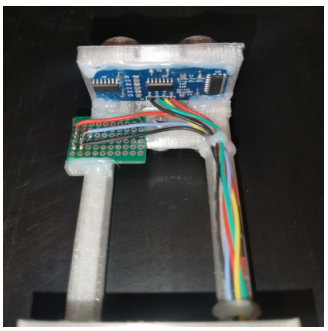


Рис. В.23: Датчик расстояния

В заключение сборки робота на оси электродвигателей одеваются колеса, которые для увеличения сцепления с трассой оклеиваются тонким двухсторонним скотчем. Окончательный вид робота представлен на рисунке В.24.

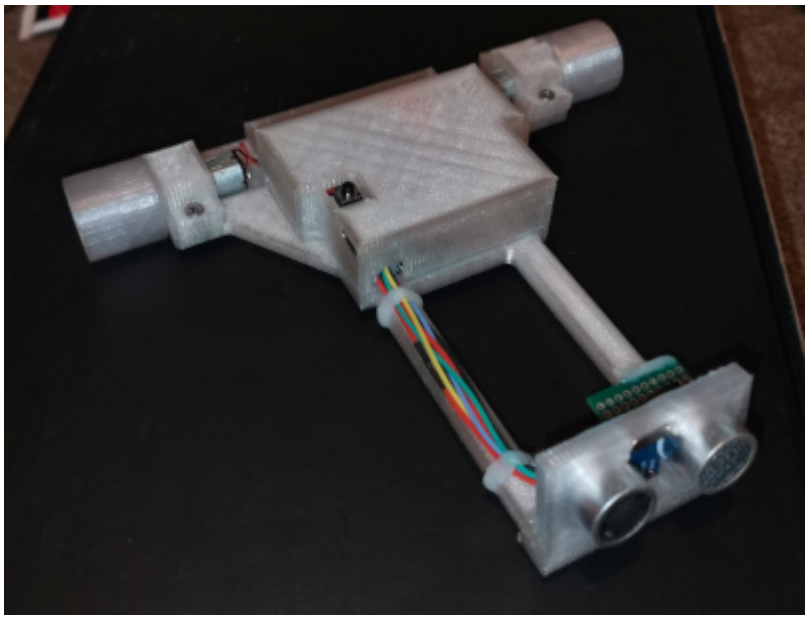


Рис. В.24: Окончательный вид робота

Питание робота осуществляется от LiPo 2S аккумулятора емкостью 400 mAh.

Приложение Г

Прошивка файловой системы робота

Для работы программ этого пособия рекомендуется в каталог /flash робота поместить содержимое архива flash_v3.zip (рисунок Г.1), который можно скачать по адресу http://www.dec.su/upload/wysiwyg/flash_v3.zip.

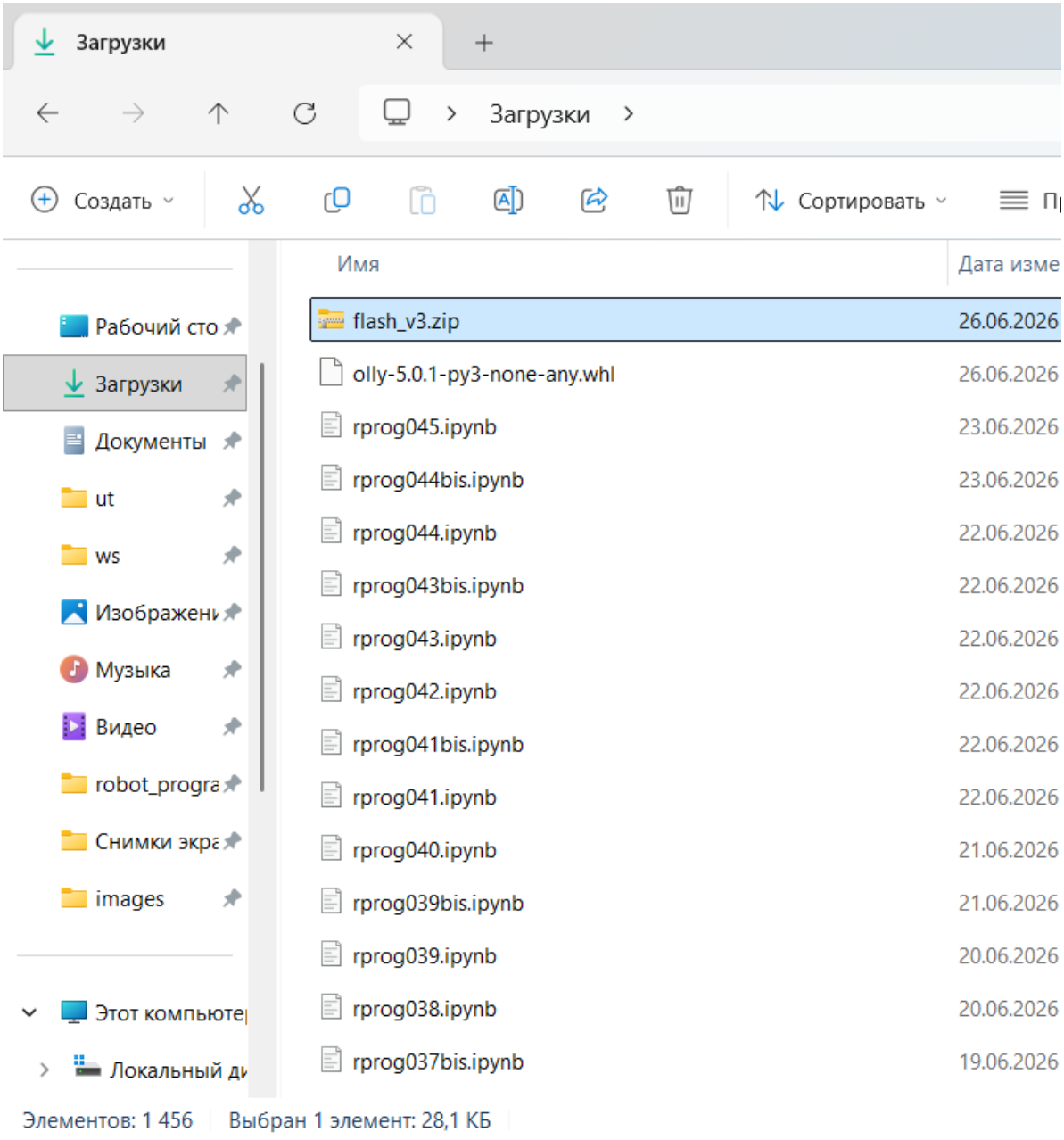


Рис. Г.1: Архив flash_v3.zip

Скачав архив следует его открыть нажав на нем правой кнопкой мыши и в всплывающем меню выбрать пункт Открыть (рисунок Г.2).

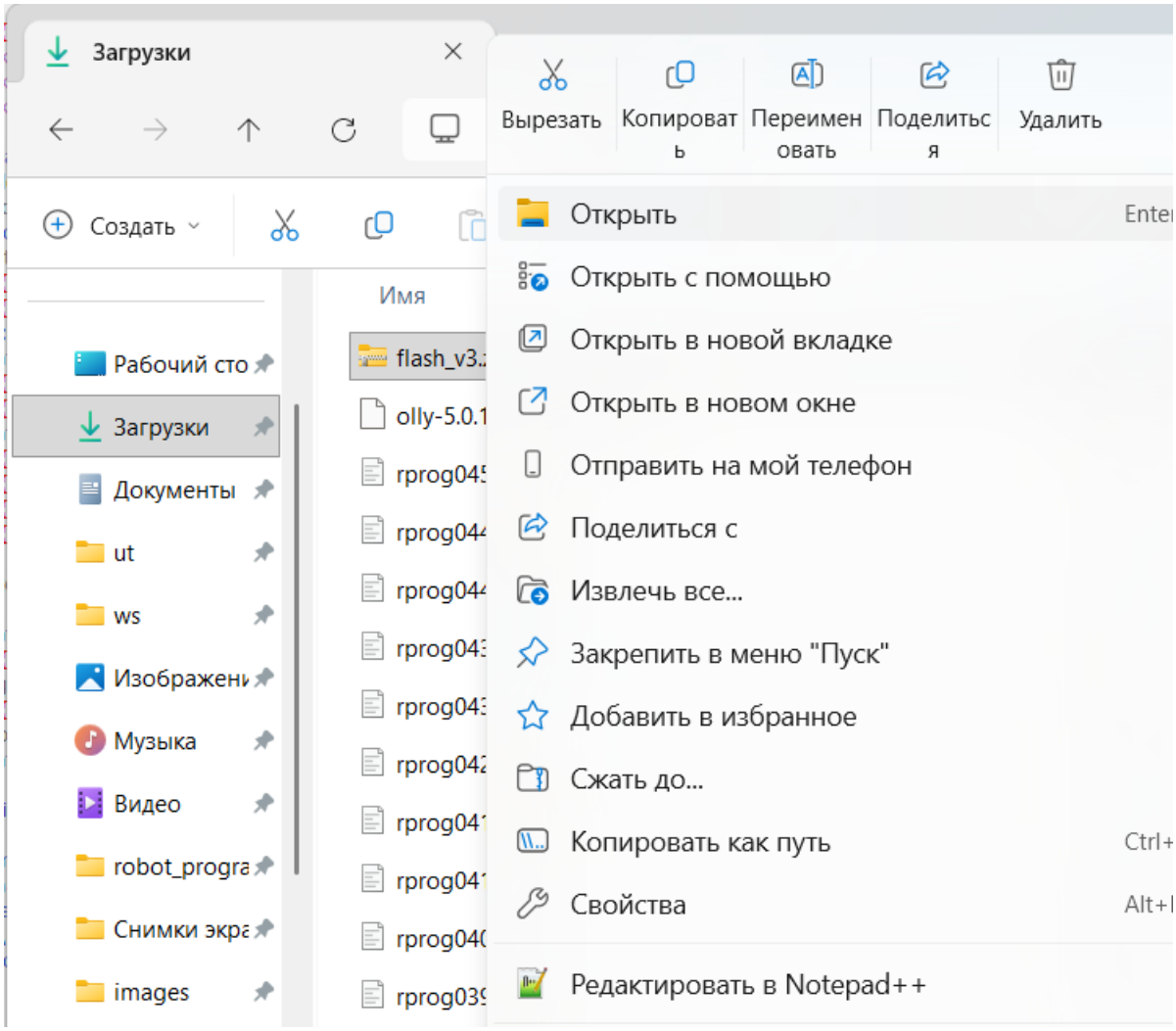


Рис. Г.2: Открытие архива flash_v3.zip

В открывшемся архиве следует выбрать выделив все находящиеся в нем каталоги и файлы (рисунок Г.3).

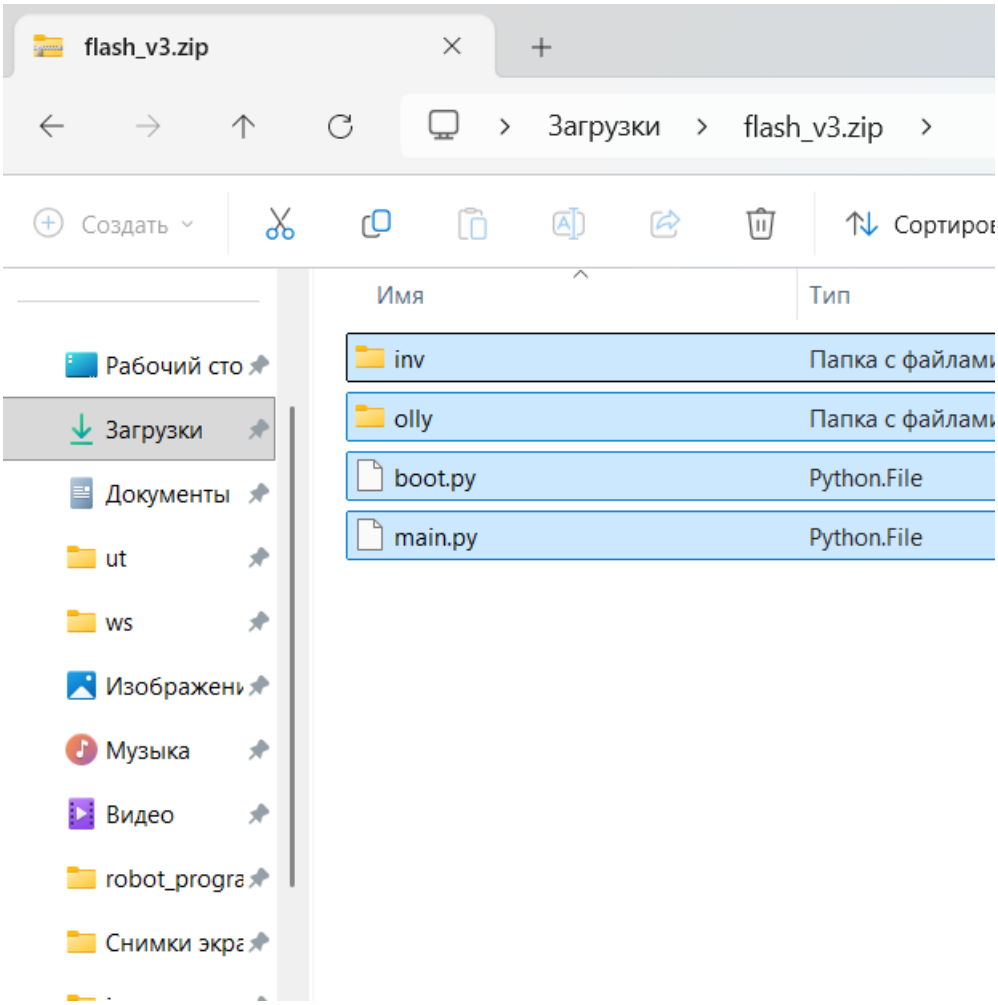


Рис. Г.3: Выбор необходимых элементов в архиве

Нажать правой кнопкой мыши по выделенным файлам и в появившемся меню выбрать пункт Копировать (рисунок Г.4).

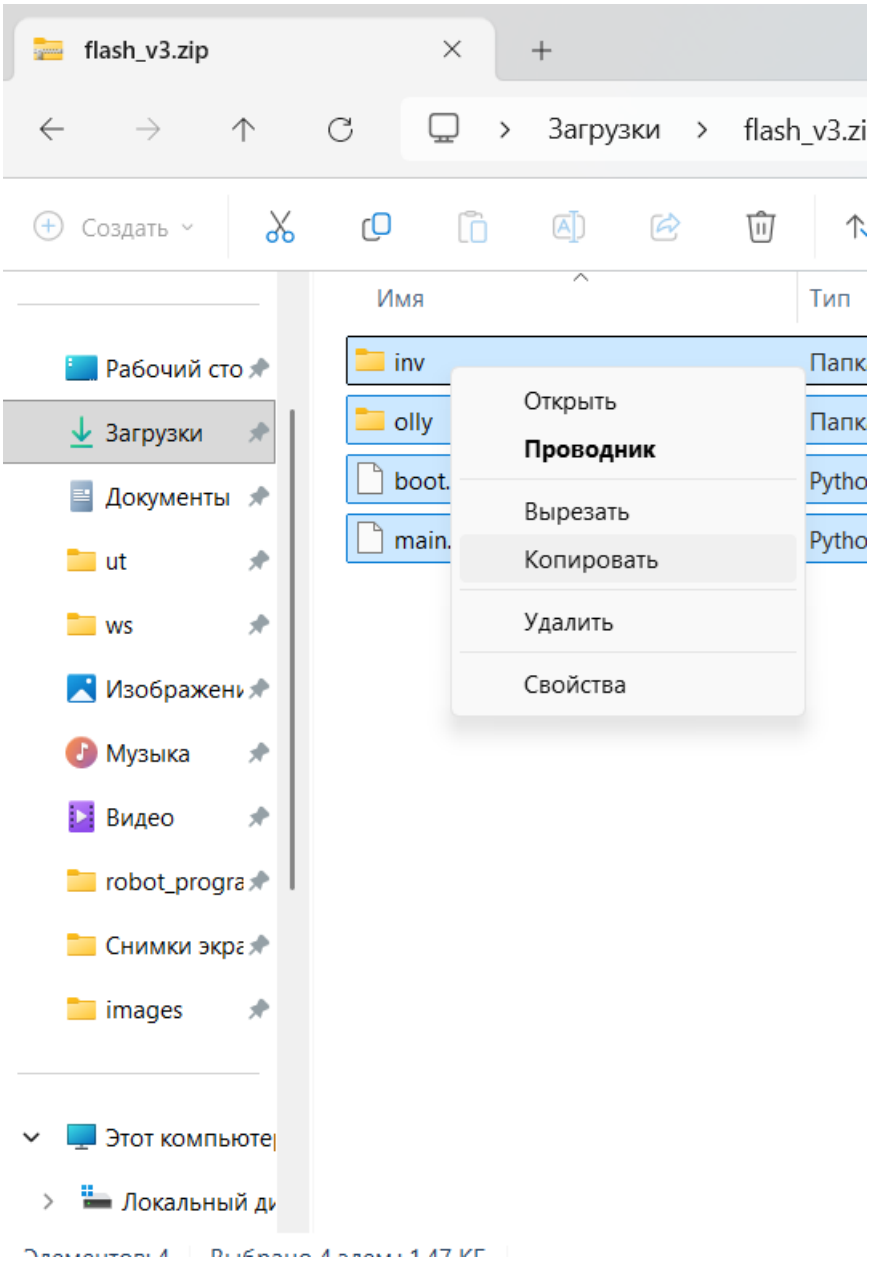


Рис. Г.4: Копирование содержимого архива

Далее необходимо вставить скопированные файлы в каталог, созданный в параграфе 1.2 (рисунок 1.10). Для этого в указанном каталоге следует нажать правой кнопкой мыши и в появившемся всплывающем меню нажать кнопку Вставить (рисунок Г.5).

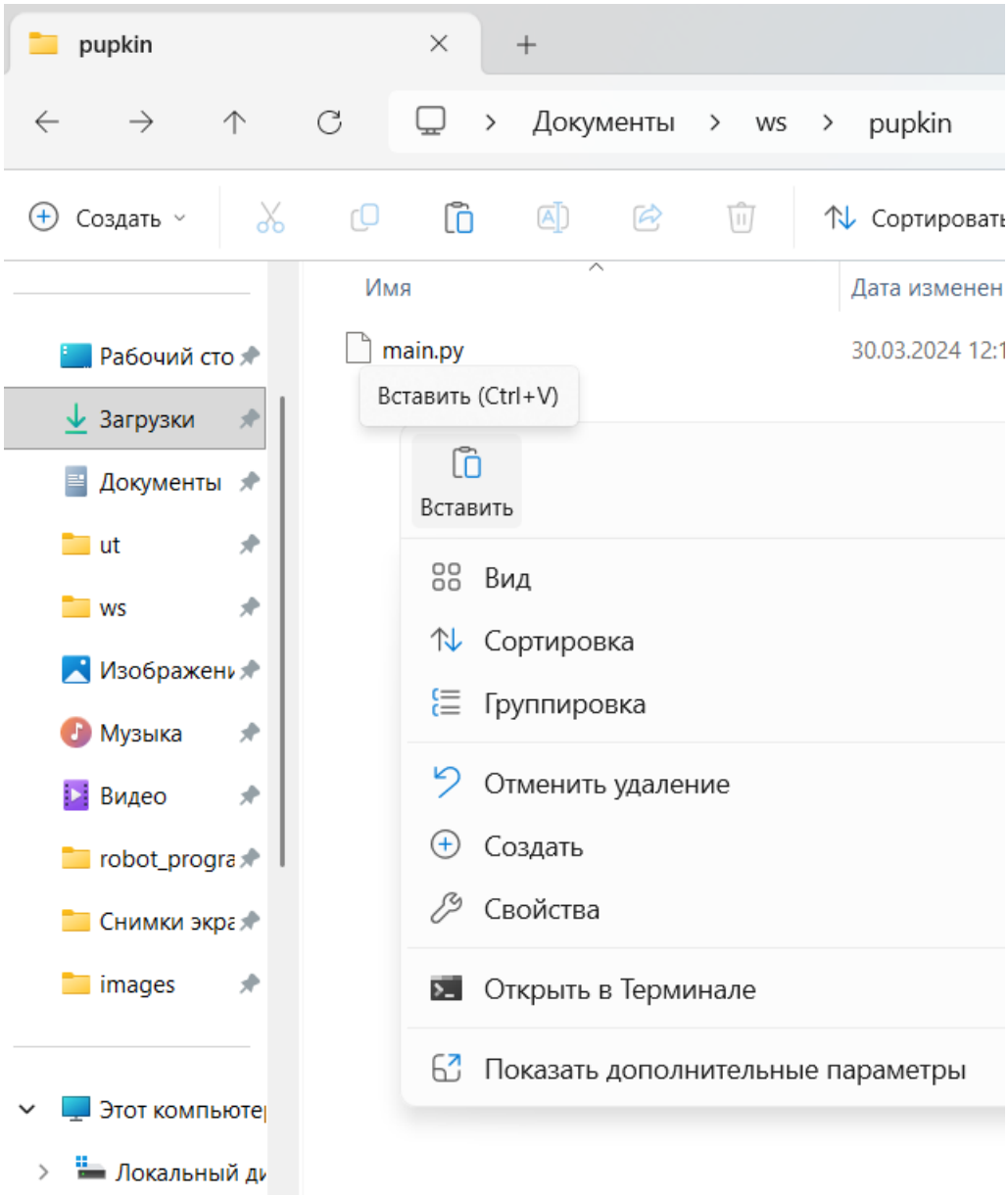


Рис. Г.5: Вставка содержимого архива в свой рабочий каталог

В результате необходимые файлы появятся в этом каталоге (рисунок Г.6).

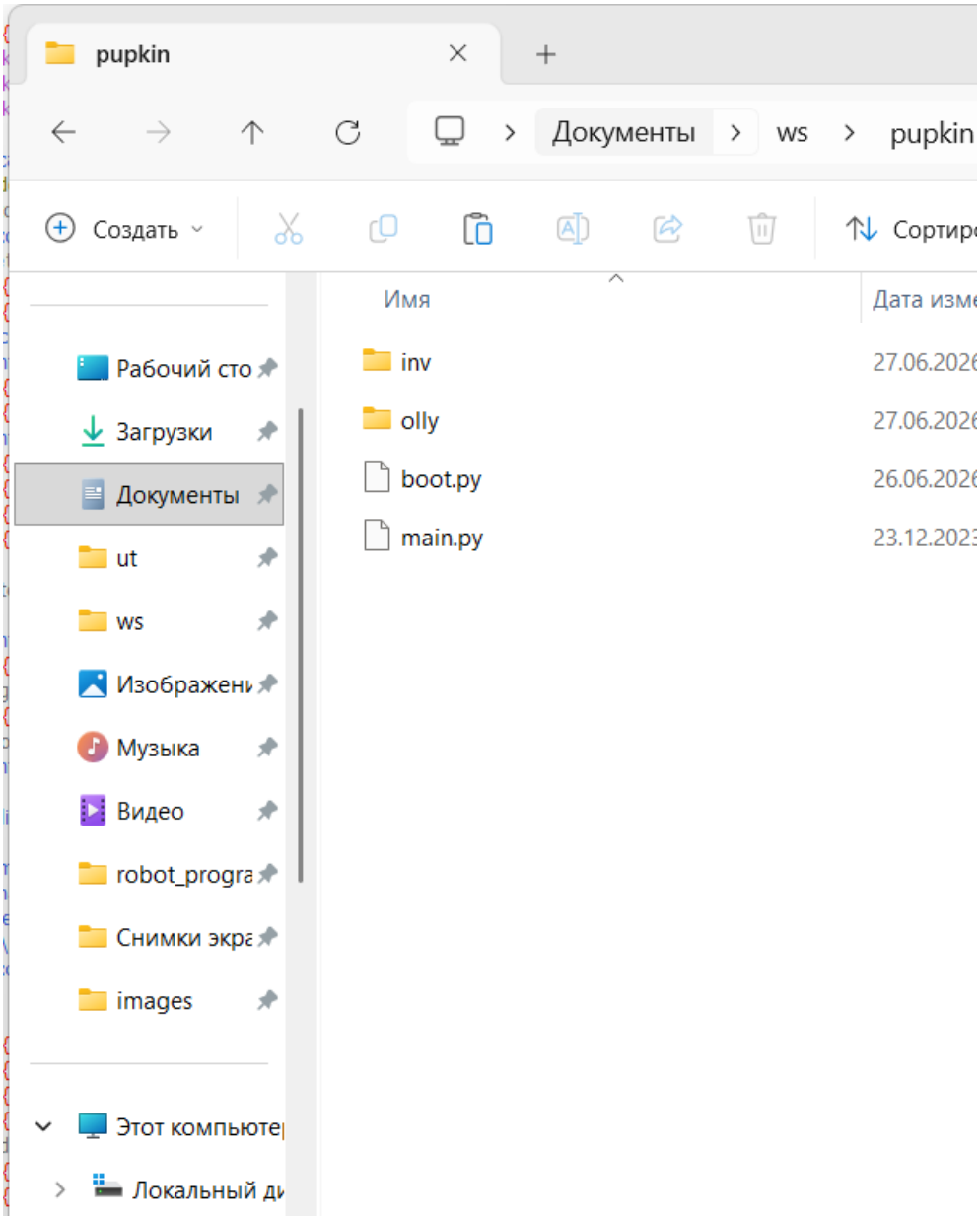


Рис. Г.6: Содержимое своего рабочего каталога

На следующем шаге необходимо подключить робота к компьютеру с помощью USB кабеля и выбрать соответствующий COM порт в настройках интегрированной среды разработки, как описано в § 1.2 (рисунок Г.7).

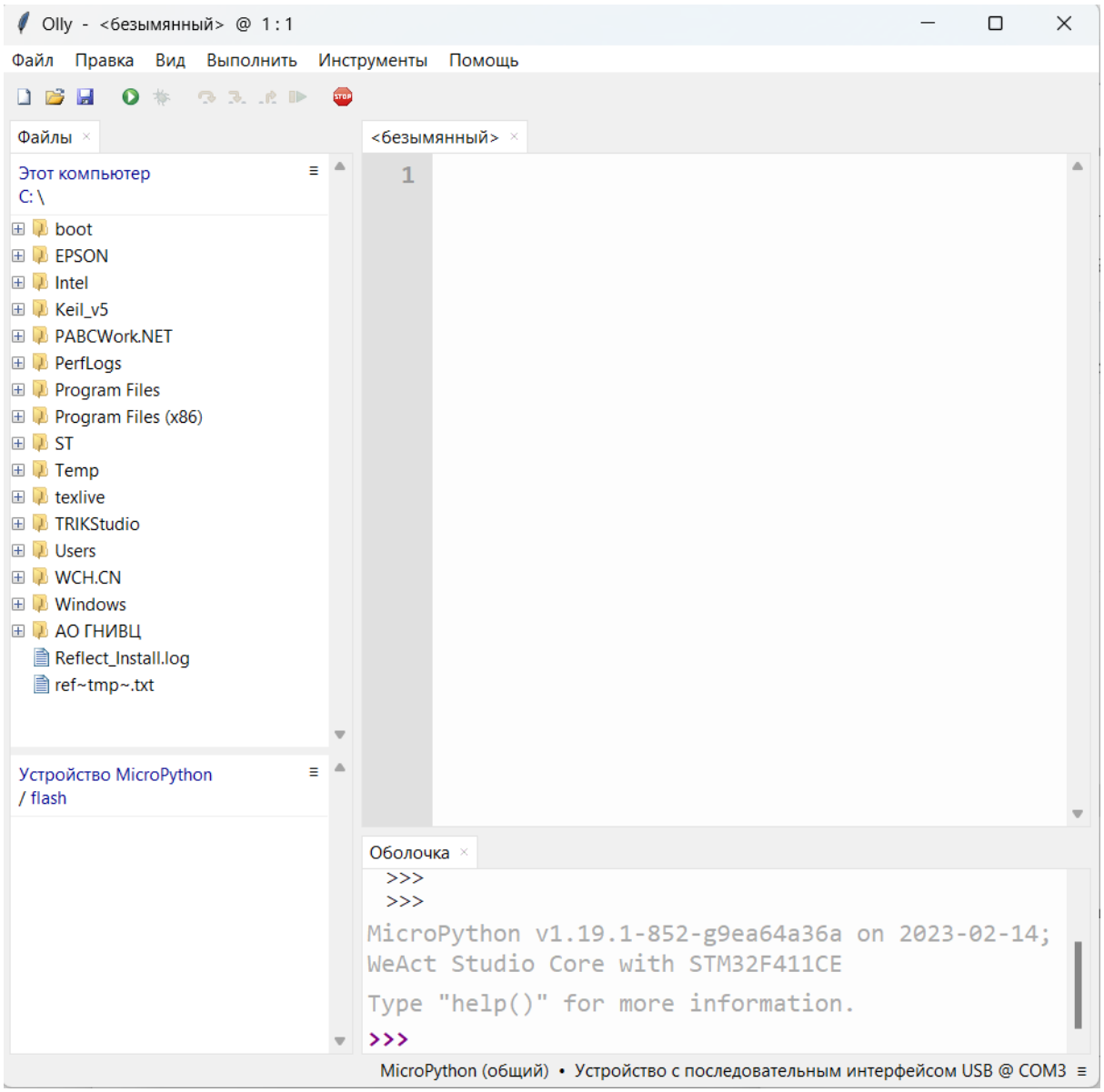


Рис. Г.7: Подключение робота к IDE Olly

Далее необходимо отформатировать внутреннюю память робота под файловую систему littlefs для чего в панели оболочки надо ввести последовательность команд как на рисунке Г.8. после чего нажать клавишу Enter.

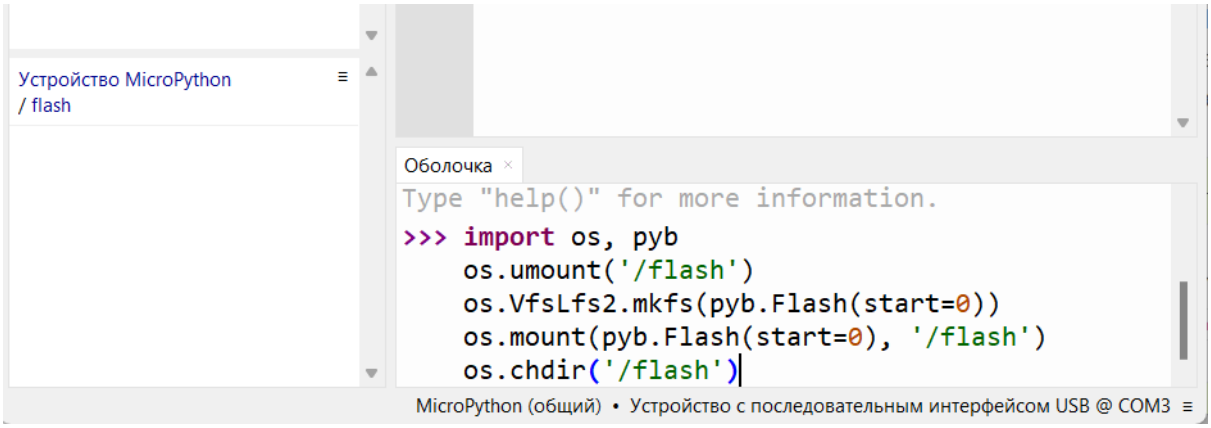


Рис. Г.8: Форматирование внутренней памяти робота

На рисунке Г.9 изображен результат выполнения форматирования.

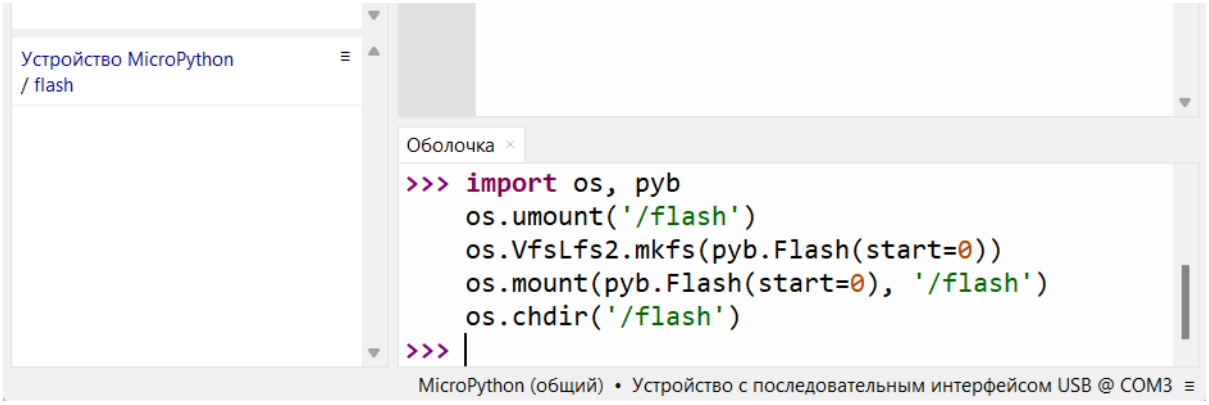


Рис. Г.9: Результат выполнения форматирования

Затем в верхней части панели Файлы следует открыть каталог (рисунок Г.6) с содержимым архива flash_v3.zip (рисунок Г.10).

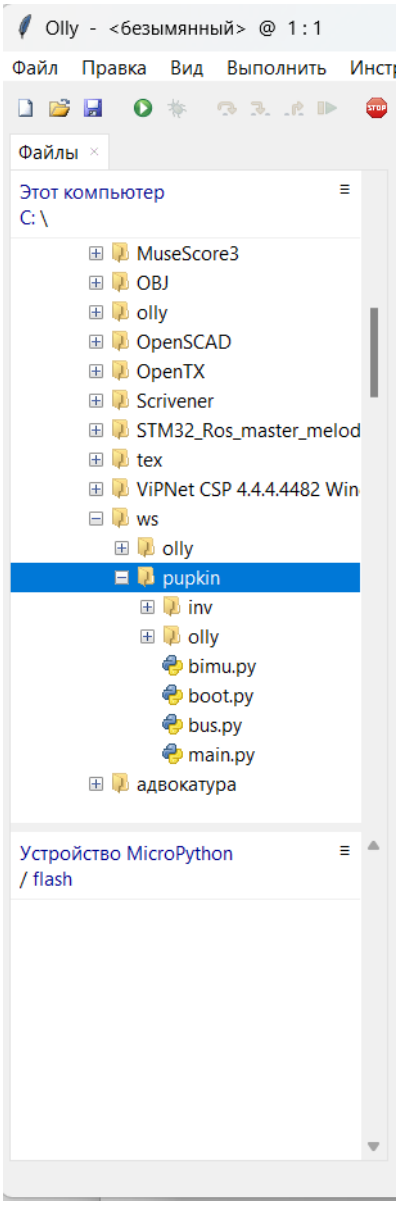


Рис. Г.10: Каталог с содержимым архива flash_v3.zip

Далее необходимо каталоги inv и olly, а также файлы boot.py и main.py последовательно загрузить на робота в каталог /flash. Для этого каждый из указанных каталогов и файлов необходимо последовательно выделить мышью, после чего нажать на нем правой кнопкой мыши и в появившемся всплывающем меню выбрать пункт Выгрузить в /flash (рисунок Г.11).

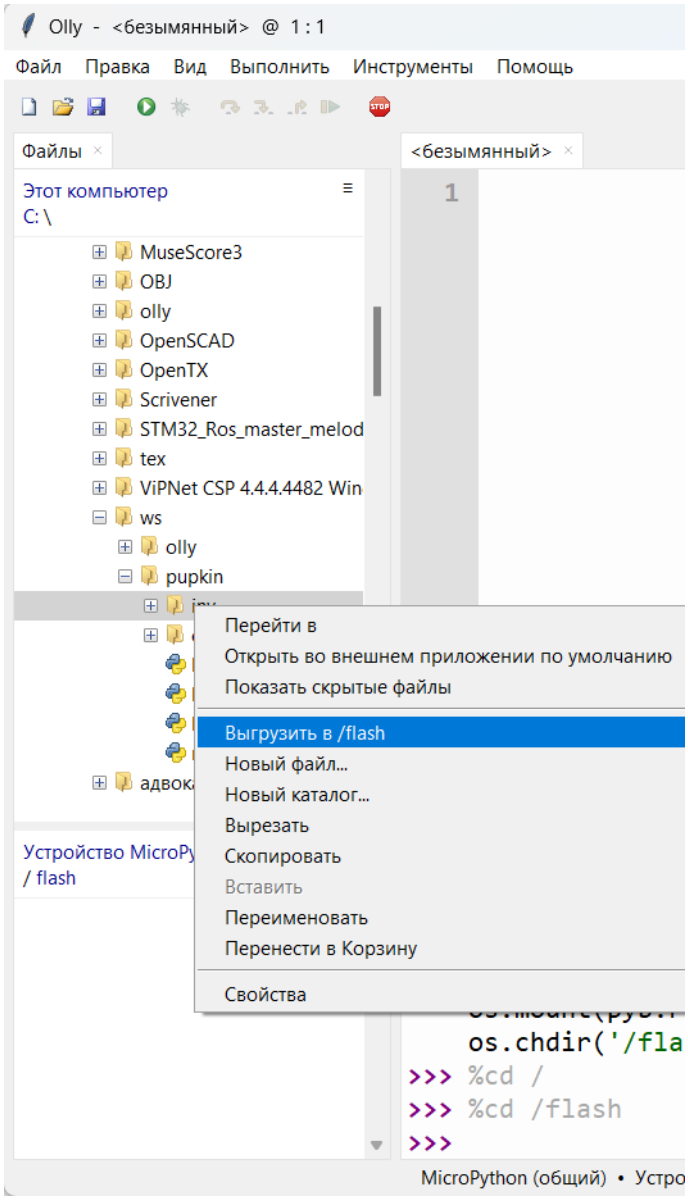


Рис. Г.11: Загрузка каталога inv на робота

Результат загрузки каталога inv на робота изображен на рисунке Г.12.

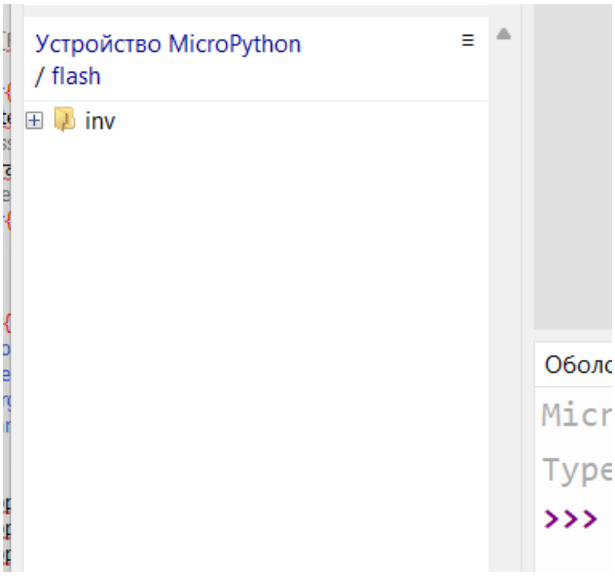


Рис. Г.12: Результат загрузки каталога inv на робота

Результат загрузки всего содержимого архива flash_v3.zip на робота изображен на рисунке Г.13.

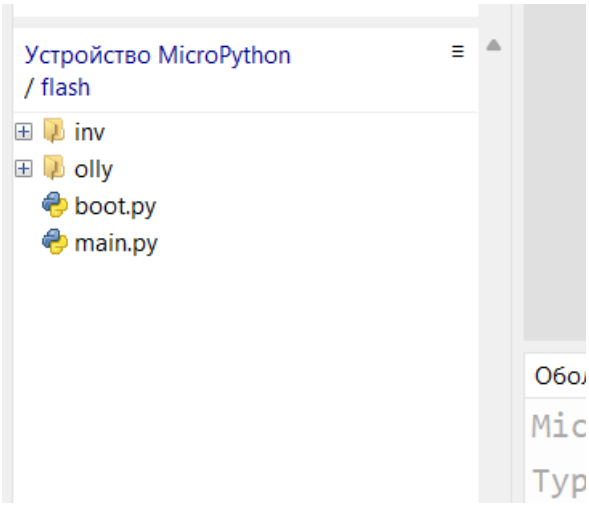


Рис. Г.13: Результат загрузки содержимого архива flash_v3.zip на робота

Приложение Д

Установка Olly

В этом приложении описан процесс установки среды разработки Olly на компьютер под управлением Windows. Рассмотрение процесса установки Olly под Linux выходит за рамки этой книги, хотя Olly будет работать и там.

Olly устанавливается на предварительно установленную среду Python. Поэтому первоначально необходимо скачать инсталляционный пакет последней версии Python с сайта pyton.org (рисунок Д.1).

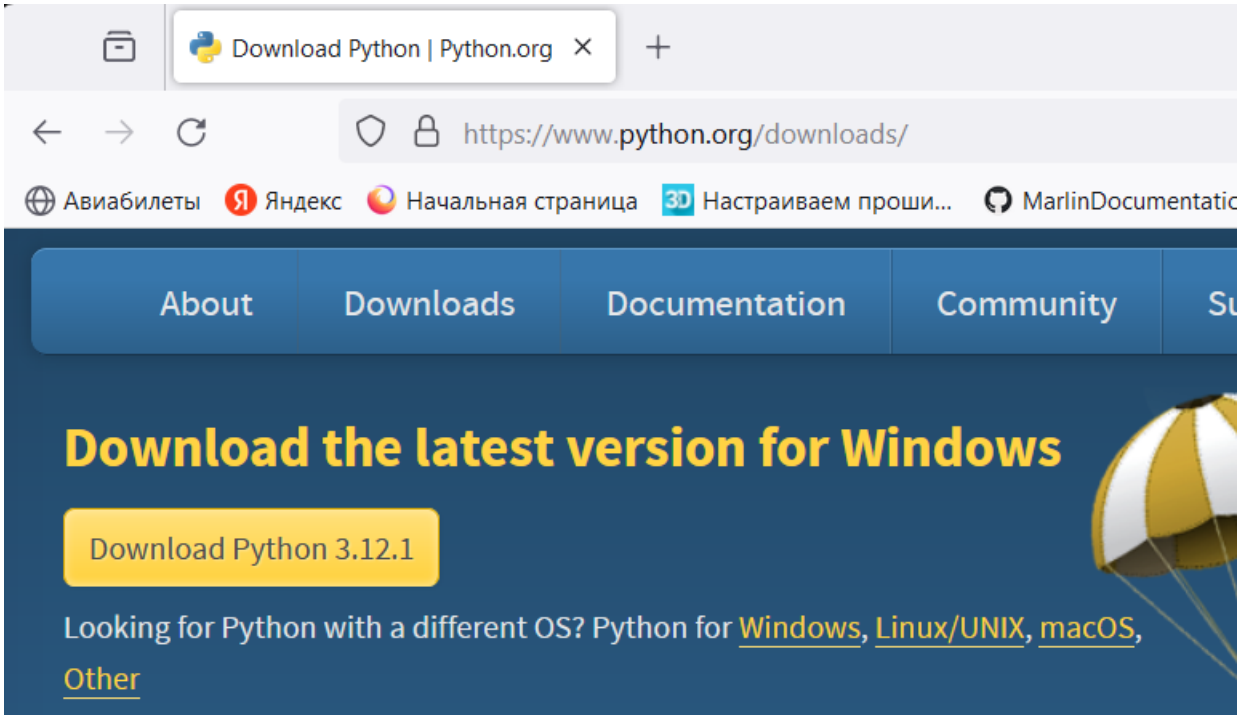


Рис. Д.1: Скачивание инсталляционного пакета Python

После окончания процесса скачивания (рисунок Д.2)

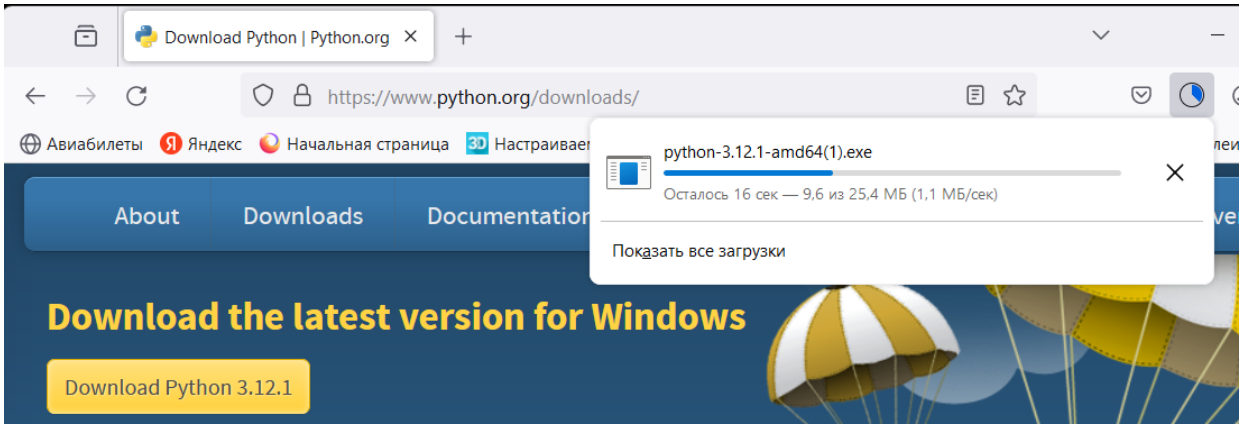


Рис. Д.2: Процесс скачивания инсталляционного пакета Python

следует открыть проводник и перейти в каталог Загрузки, а затем найти скачанный файл (рисунок Д.3) (в приводимом примере это python-3.12.1-amd64)

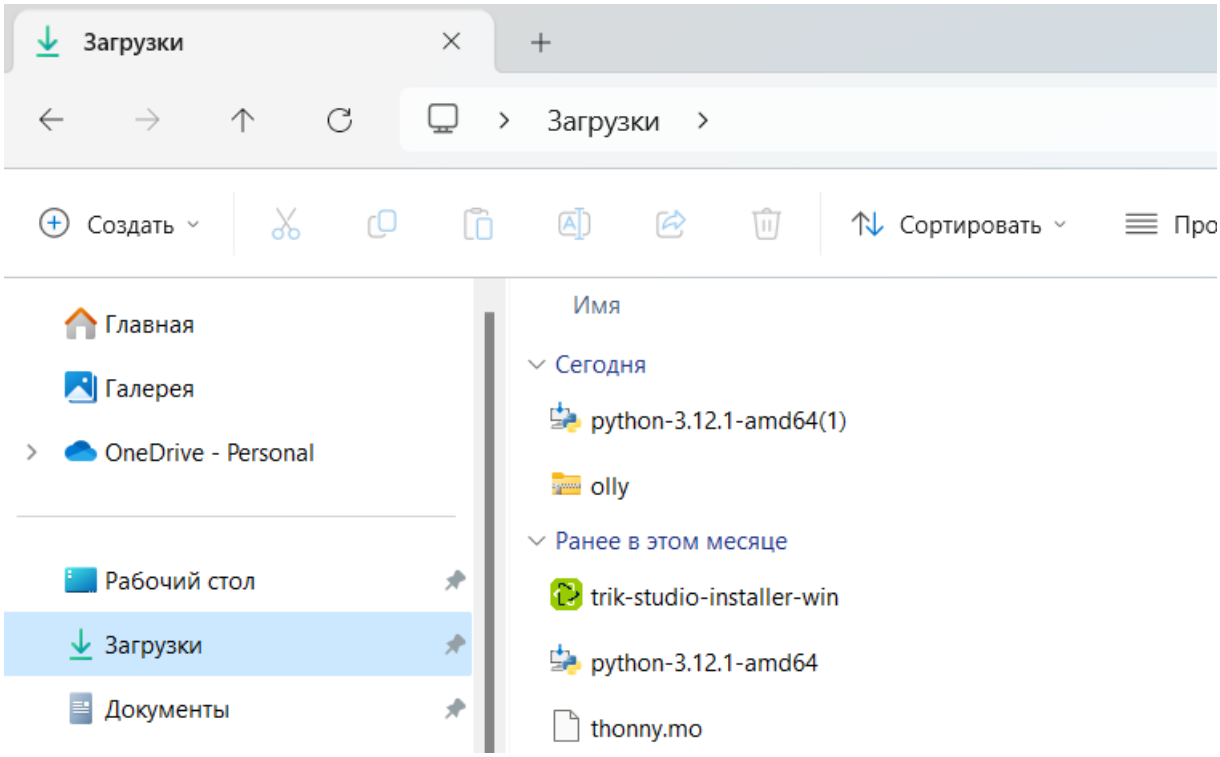


Рис. Д.3: Содержимое каталога Загрузки

и запустить его, после чего начнется процесс установки Python (рисунок Д.4).

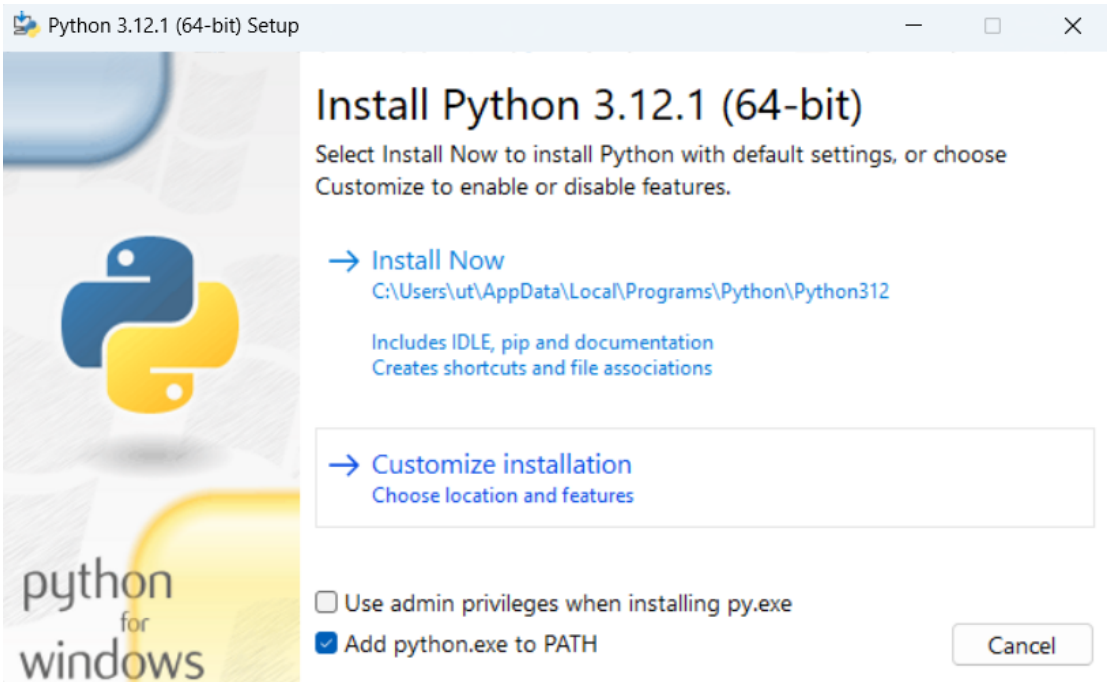


Рис. Д.4: Начало установки Python

На этом шаге следует отметить пункт Add python.exe to PATH (согласно рисунку) и выбрать вариант Customize istallation.

На следующем шаге следует выбрать опции как показано на рисунке Д.5 и нажать кнопку Next.

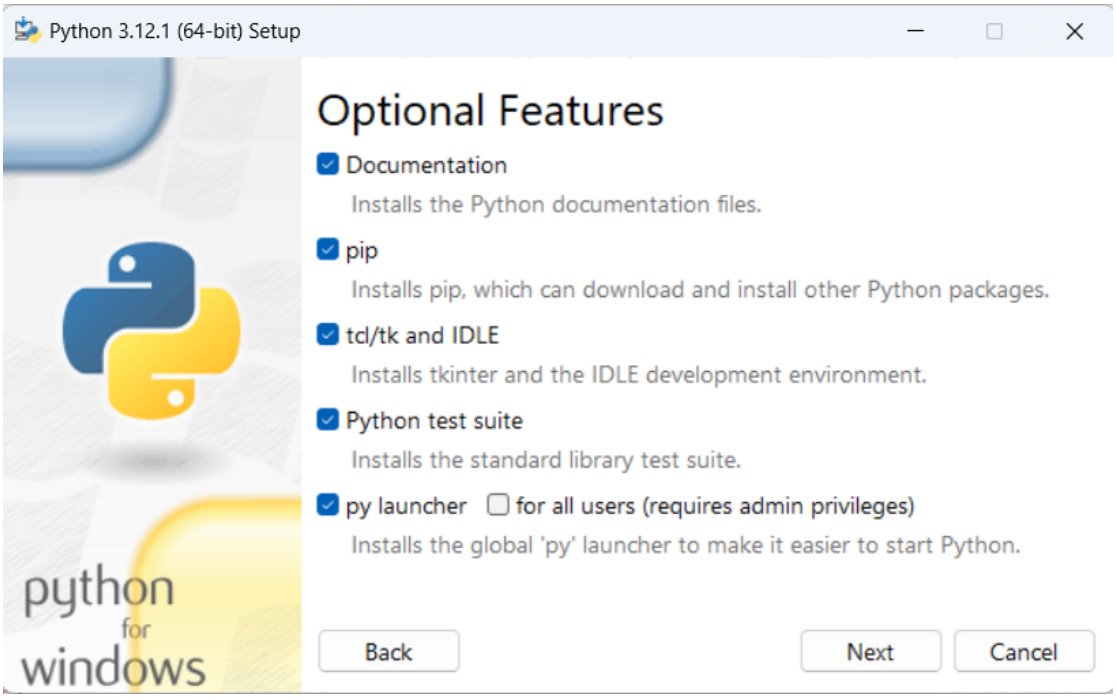


Рис. Д.5: Выбор опций установки

Далее выбираются опции согласно рисунку Д.6 и нажимается кнопка Install.

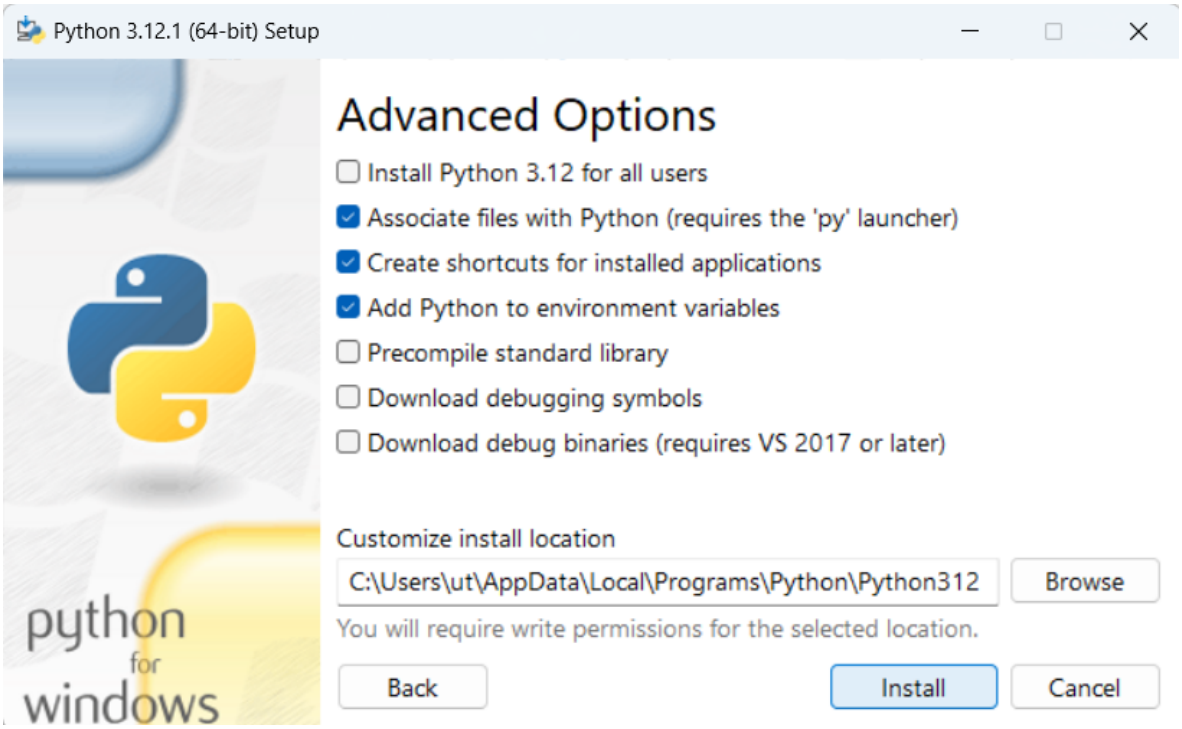


Рис. Д.6: Выбор дополнительных опций установки

После чего начинается процесс установки Python (рисунок Д.7).

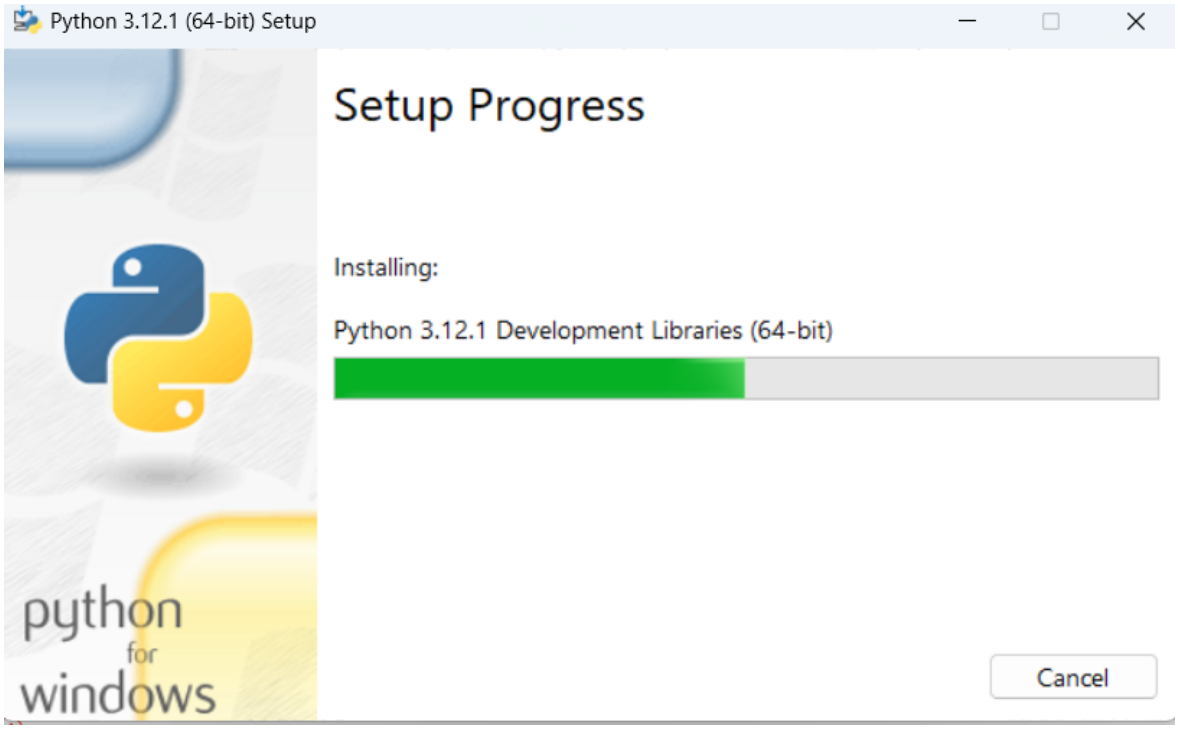


Рис. Д.7: Процесс установки

После успешной установки Python (рисунок Д.8) компьютер следует перезапустить.

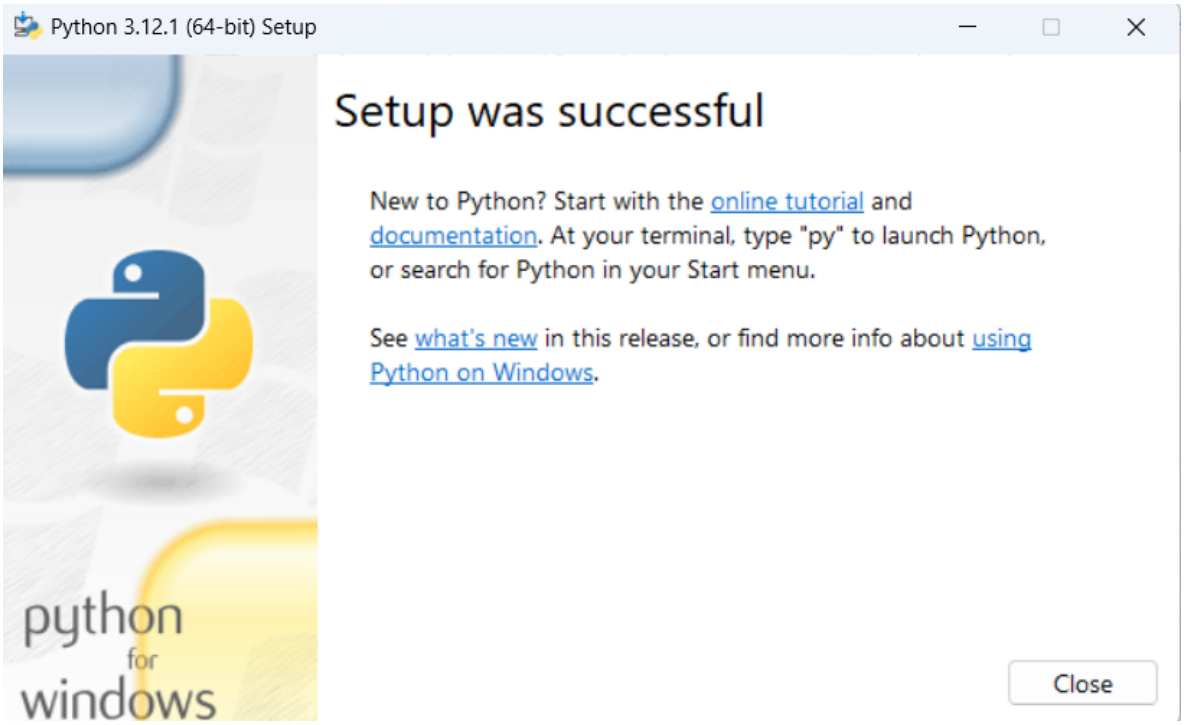


Рис. Д.8: Окончание установки Python

Далее необходимо скачать whl пакет интегрированной среды разработки Olly по адресу http://www.dec.su/upload/wysiwyg/olly-5.0.1-py3-none-any.whl, и в проводнике в папке Загрузки найти скачанный архив (рисунок Д.9).

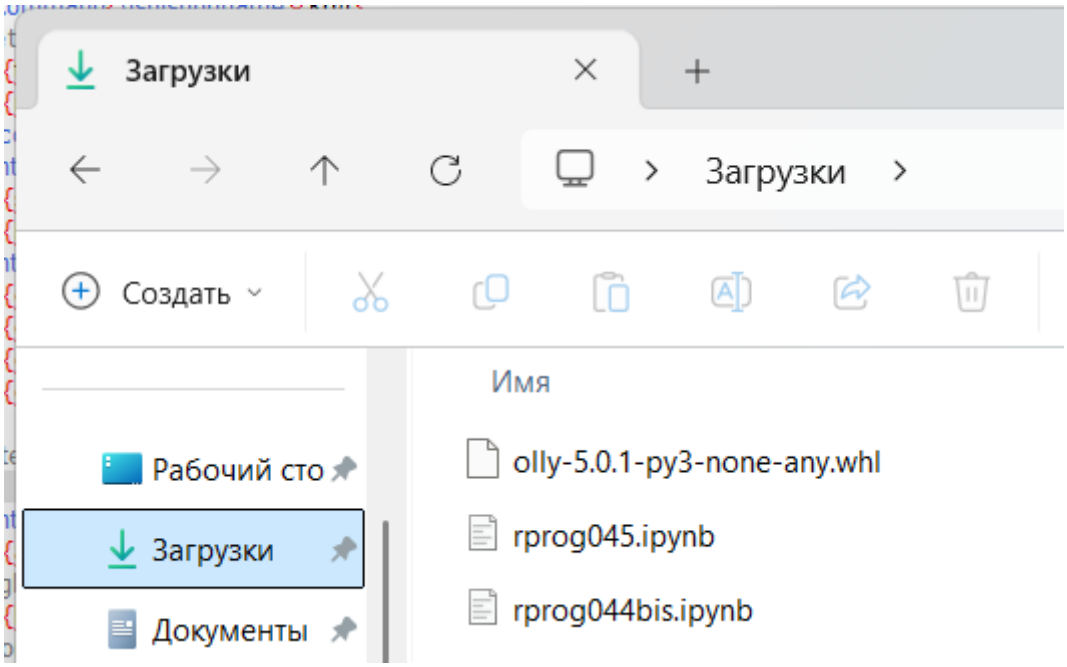


Рис. Д.9: whl пакет в папке Загрузки

Далее в проводнике следует нажать на скачанном пакете правой кнопкой мыши и в открывшемся меню выбрать пункт “Копировать как путь” (рисунок Д.10).

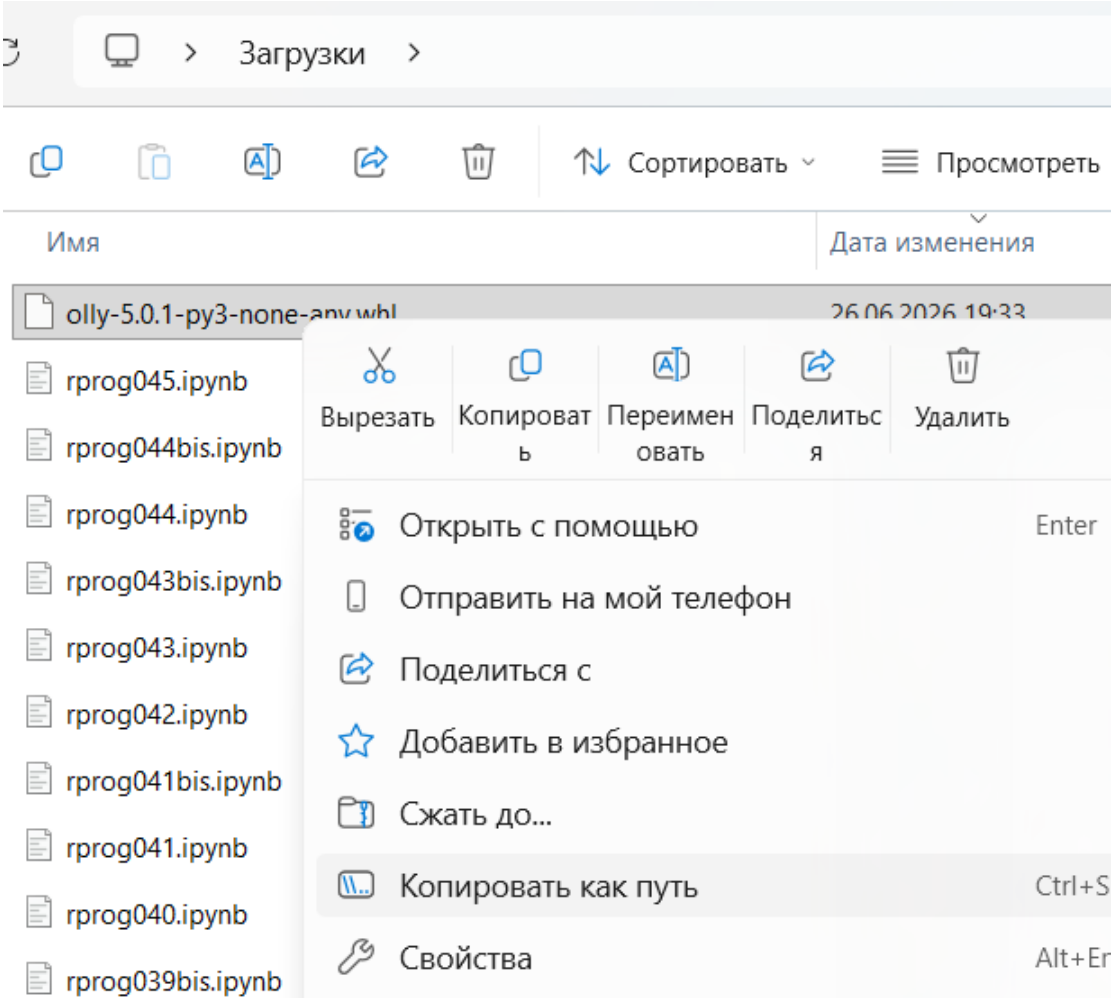


Рис. Д.10: Копирование пути whl пакета

Затем следует нажать комбинацию клавиш “Windows + R”, в строке ввода всплывшего окна набрать команду “pip install” и вставить скопированный путь с помощью комбинации клавиш “Ctrl + V”, затем нажать кнопку “OK” (рисунок Д.11),

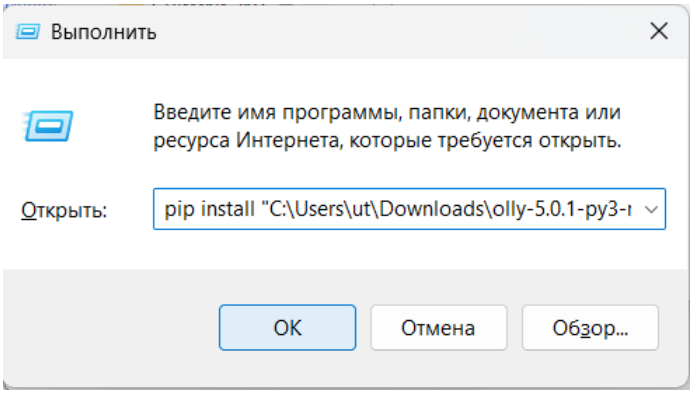


Рис. Д.11: Установка whl пакета

что приведет к запуску процесса установки интегрированной среды разработки Olly (рисунок Д.12).

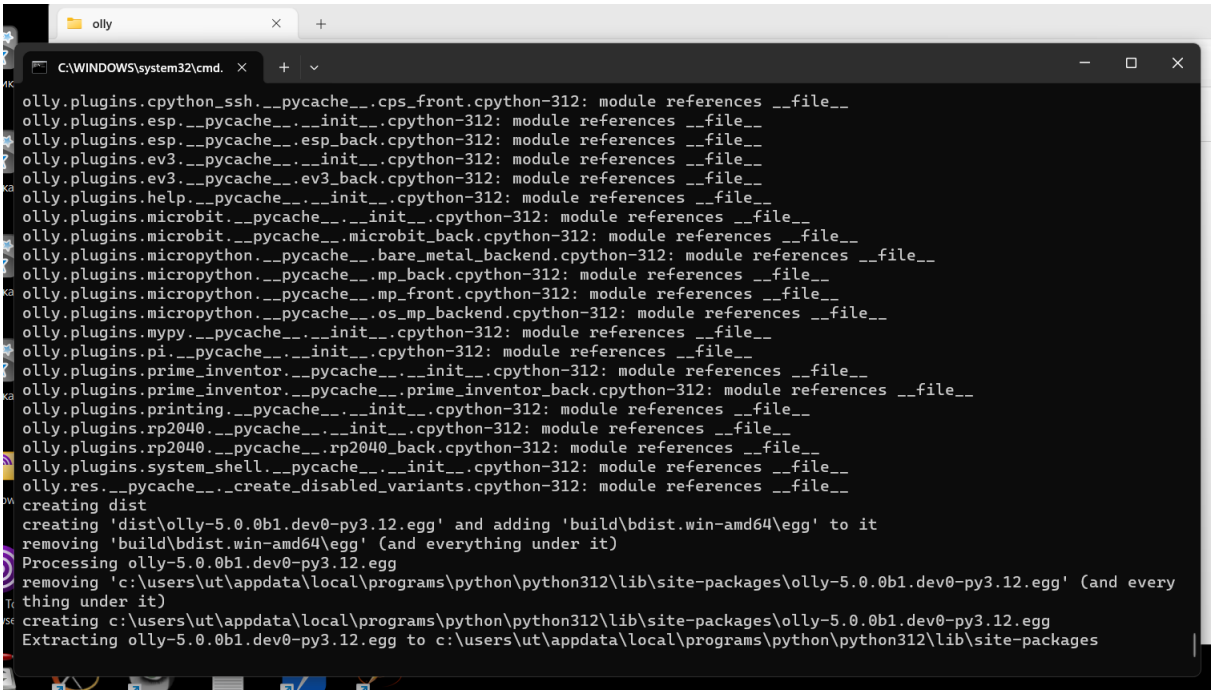


Рис. Д.12: Процесс установки интегрированной среды разработки Olly

После окончания процесса установки IDE Olly запустить ее можно выполнив следующие операции: нажать комбинацию клавиш “Windows + R” и в строке ввода всплывшего окна набрать команду “olly”, затем нажать кнопку “OK” , рисунок Д.13).

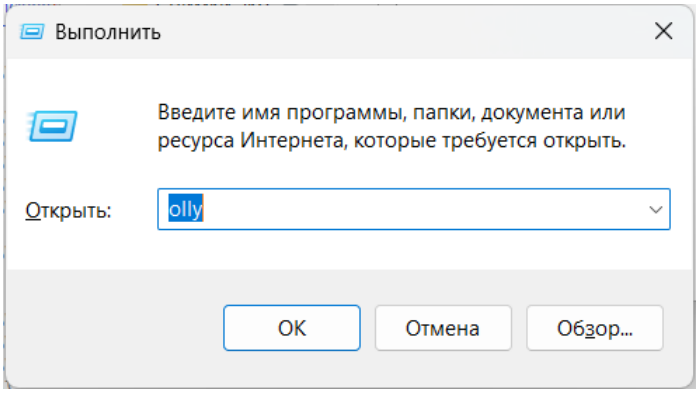


Рис. Д.13: Запуск IDE Olly

При первом запуске желательно выбрать русский язык и тип пользовательского интерфейса:

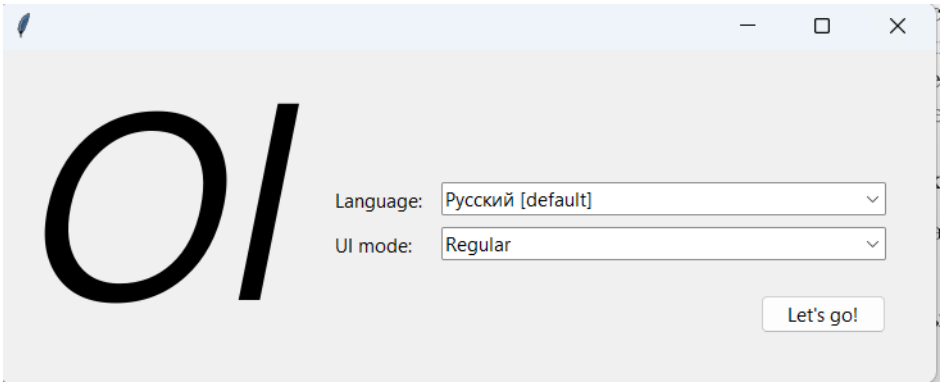


Рис. Д.14: Выбор языка оболочки и типа интерфейса

Нажатие на кнопку “Lat’s go” приведет к запуску интегрированной среды разработки Olly.

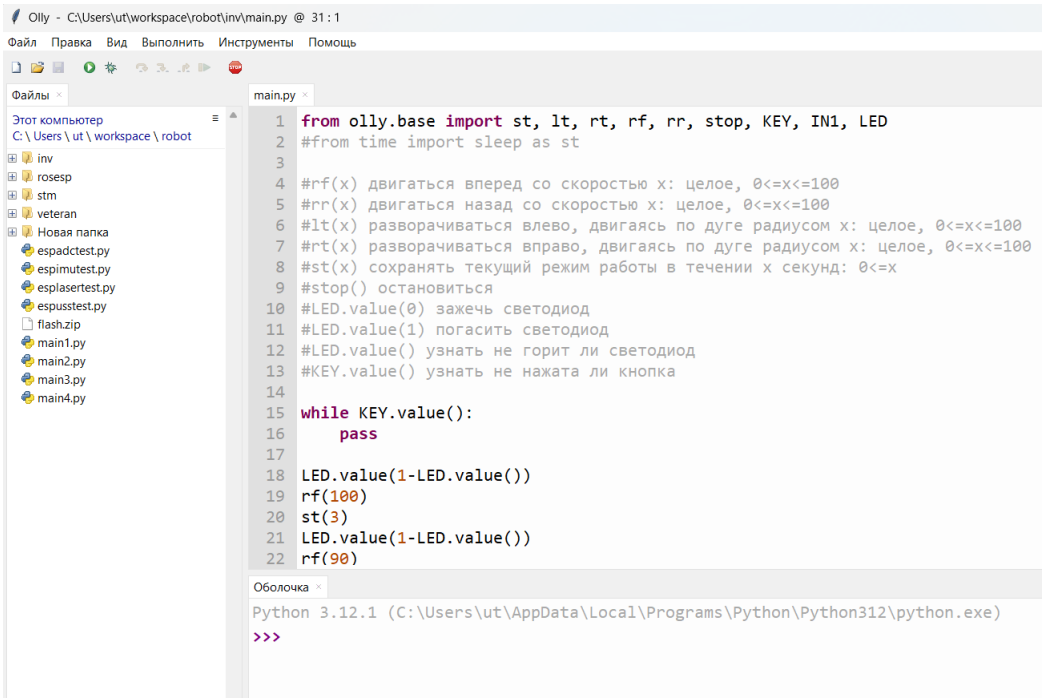


Рис. Д.15: Запущенная среда разработки Olly

Приложение Е

base.py

```
#Copyright (C) 2023 Jury Polikarpov
#na_5050@bk.ru www.dec.su +79516739548
from pyb import Pin, Timer, ADC
from time import sleep

def st(s):
    sleep(s)

IN1  = ADC(Pin('PA1'))
LED = Pin('PC13',Pin.OUT,Pin.PULL_NONE)
KEY = Pin('PB10',Pin.IN,Pin.PULL_UP)
TIM4 = Timer(4, freq=10000)
TIM4_CH1 = Pin('PB6',Pin.PULL_NONE)
TIM4_CH1 = TIM4.channel(1, Timer.PWM,
   pin=TIM4_CH1,pulse_width_percent = 0)
TIM4_CH2 = Pin('PB7',Pin.PULL_NONE)
TIM4_CH2 = TIM4.channel(2, Timer.PWM,
   pin=TIM4_CH2,pulse_width_percent = 0)
TIM4_CH3 = Pin('PB8',Pin.PULL_NONE)
TIM4_CH3 = TIM4.channel(3, Timer.PWM,
   pin=TIM4_CH3,pulse_width_percent = 0)
TIM4_CH4 = Pin('PB9',Pin.PULL_NONE)
TIM4_CH4 = TIM4.channel(4, Timer.PWM,
   pin=TIM4_CH4,pulse_width_percent = 0)
#TIM4_CH1.pulse_width_percent(50)
#TIM4_CH2.pulse_width(int(TIM4.period()/2))
def stop():
    TIM4_CH1.pulse_width_percent(0)
    TIM4_CH2.pulse_width_percent(0)
    TIM4_CH3.pulse_width_percent(0)
    TIM4_CH4.pulse_width_percent(0)
lmfi=1
rmfi=1
plmfi=1
prmfi=1
def rmf(n):
    global rmfi, lmfi, plmfi, prmfi
    TIM4_CH1.pulse_width_percent(0)
    TIM4_CH2.pulse_width_percent(n)
    rmfi=1
def rmr(n):
    global rmfi, lmfi, plmfi, prmfi
```

```
39      TIM4_CH2.pulse_width_percent(0)
40      TIM4_CH1.pulse_width_percent(n)
41      rmfi=-1
42  def lmr(n):
43      global rmfi, lmfi, plmfi, prmfi
44      TIM4_CH3.pulse_width_percent(0)
45      TIM4_CH4.pulse_width_percent(n)
46      lmfi=-1
47  def lmf(n):
48      global rmfi, lmfi, plmfi, prmfi
49      TIM4_CH4.pulse_width_percent(0)
50      TIM4_CH3.pulse_width_percent(n)
51      lmfi=1
52  def nrml(n):
53      if (n>=0)and(n<=100):
54          return int(n)
55      elif (n<0):
56          return 0
57      elif (n>100):
58          return 100
59      return (n)
60  def rf(n):
61      global rmfi, lmfi, plmfi, prmfi
62      n=nrml(n)
63      plmfi=1
64      prmfi=1
65      if (plmfi*lmfi+prmfi*rmfi)<1:
66          stop()
67          sleep(0.2)
68      rmf(n)
69      lmf(n)
70  def rr(n):
71      global rmfi, lmfi, plmfi, prmfi
72      n=nrml(n)
73      plmfi=-1
74      prmfi=-1
75      if (plmfi*lmfi+prmfi*rmfi)<1:
76          stop()
77          sleep(0.2)
78      rmr(n)
79      lmr(n)
80  def rt(n):
81      global rmfi, lmfi, plmfi, prmfi
82      n=nrml(n)
83      plmfi=1
84      if n==0:
85          prmfi=-1
86      else:
87          prmfi=1
88      if (plmfi*lmfi+prmfi*rmfi)<1:
89          stop()
90          sleep(0.2)
91      if n==0:
```

```
92          rmr(100)
93      else:
94          rmf(int(100-100/n))
95      lmf(100)
96  def lt(n):
97      global rmfi, lmfi, plmfi, prmfi
98      n=nrml(n)
99      prmfi=1
100     if n==0:
101         plmfi=-1
102     else:
103         plmfi=1
104     if (plmfi*lmfi+prmfi*rmfi)<1:
105         stop()
106         sleep(0.2)
107     if n==0:
108         lmr(100)
109     else:
110         lmf(int(100-100/n))
111     rmf(100)
```

Приложение Ж

Расширенный симулятор nbase

Разработанный расширенный симулятор представляет собой высокоточный цифровой двойник учебного робота Ветеран. Симулятор сочетает в себе физический движок (расчет движения, жестких столкновений и веерного дальномера) и аппаратный эмулятор API (команды моторов, светодиода, кнопки старта и датчиков идентичны прошивке реального микроконтроллера STM32).

1. Архитектура и структура симулятора

Симулятор поставляется в составе единого скомпилированного Wheel-пакета (.whl) IDE Olly и состоит из трех ключевых модулей и файла конфигурации сцены:

- olly.nbase — графическое и физическое ядро симулятора. Отвечает за независимый двухпоточный рендеринг Pygame, симуляцию корпуса-трапеции робота (база 14 см), обработку коллизий всей плоскостью бампера, опрос клавиатуры и аналогового ИК-датчика линии.
- olly.bus — модуль цифровой периферии. Содержит функцию ультразвукового дальномера dstnc().
- olly.bimu — модуль инерциальной навигационной системы (ИМУ). Моделирует работу гироскопа, акселерометра и датчика температуры платы.
- default_scene.yaml — файл текстовой разметки соревновательного полигона по умолчанию.

2. Технические и физические характеристики

Симулятор откалиброван под параметры реального робота и соревновательные регламенты:

- Масштаб поля: 4 пикселя симулятора = 1 реальный сантиметр поля. Окно имеет размер 800x600 пикселей (200x150 см в реальности).
- Физическая скорость (MAX_SPEED_CMS): 3.5 см/с при мощности моторов 100. Робот движется плавно, давая датчикам время гарантированно отработать трассу.
- Ширина черной линии: 5 см (20 пикселей).
- Позиция ИК-датчика линии (IN1): Смещен на левый угол переднего бампера. Зона считывания имеет плавное аналоговое размытие шириной 1 см.
- Ультразвуковой дальномер (dstnc()): Моделирует физический конус обзора в 15° с помощью плотного веера из 13 лучей, что гарантирует обнаружение объектов любого размера без пропусков между лучами.

3. Легенда символов карты (.yaml) и слои

Карта соревновательного полигона генерируется автоматически при старте программы на основе текстовой матрицы в YAML-файле. Симулятор поддерживает послойный рендеринг (линии сглаживаются в первую очередь, а физические объекты водружаются поверх них):

- Пробел — Чистый белый пол полигона (ИК-датчик выдает значение 3000).
- L, R, K, S, T — Пять независимых слоев черной линии. Каждый слой сглаживается сплайнами Катмулла-Рома изолированно от других, что позволяет строить идеальные перекрестки, развилки и разомкнутые линии любой сложности без деформаций на изгибах. На черной линии ИК-датчик выдает 1000, на границе — около 2000.
- # — Низкий защитный бортик (светло-серый, цвет 200, 200, 200). Робот физически упирается в него бампером, но ультразвуковой датчик пролетает выше и его не видит.
- W — Высокое жесткое препятствие / стена (темно-серый куб, цвет 100, 100, 100). Блокирует движение робота, а также обнаруживается УЗ-дальномером.
- B — Подвижный круглый блок / бочка (оранжевый диск диаметром 5 см). Обладает физикой твердого тела, обнаруживается УЗ-дальномером.

4. Аппаратные ловушки и несовершенства датчиков

Симулятор специально эмулирует физические ограничения реального железа для обучения детей грамотному программированию:

- Фиксированный таймаут сонара (US_POLL_DELAY = 0.02): При вызове функции dstnc() поток ученика принудительно засыпает на 20 миллисекунд (0.02 сек). Это эмулирует фиксированный аппаратный цикл измерения реального датчика. Слишком частый опрос сонара затормозит движение робота по линии, поэтому школьники должны выносить дальномер в отдельный поток через модуль _thread.
- Слепая зона сонара (US_BLIND_ZONE_CM = 3.0): Если робот упирается вплотную в стену W или бочку B (ближе 3 см), датчик «слепнет» из-за слишком быстрого возврата эха. В этот момент функция dstnc() начинает выдавать хаотичный паспортный максимум — 150 см, имитируя сбой реального HC-SR04.

5. Динамическое управление сценой из кода

Платформа поддерживает динамическое изменение конфигурации поля прямо во время выполнения скрипта без редактирования YAML-файлов:

- set_initial(x_cm, y_cm, theta_deg) — Мгновенно перемещает робота на нужную стартовую позицию поля и разворачивает его под правильным углом, не перезагружая весь симулятор целиком. Координаты задаются в сантиметрах, угол — в градусах.
- add_block(x_cm, y_cm) — Возводит жесткий блок высокой стены W (размером 5x5 см) в указанных координатах. Финальный послойный рендеринг автоматически нарисует куб поверх черной линии, ИК-датчик потеряет линию (выдаст 3000), а УЗ-датчик начнет видеть преграду.

- add_barrel(x_cm, y_cm) — Сбрасывает на поле подвижную круглую оранжевую бочку B (диаметром 5 см).Физика толкания бочек всей шириной бампера: Если робот наезжает на круглую бочку любой точкой своего широкого плоского бампера-трапеции (а не только центром), бочка плавно и реалистично катится перед ним, смещаясь строго по вектору его движения next_theta. Бочка не проваливается сквозь робота и физически застревает, если упереть её в стену W.

6. Порядок запуска и завершения

- Ожидание старта: Команда while KEY.value(): st(0.02) удерживает робота на месте, не перегружая процессор ПК.
- Активация: Чтобы робот поехал, необходимо кликнуть мышкой внутрь открывшегося окна симулятора (передать фокус ОС) и зажать клавишу Shift (левый или правый).
- Удержание окна и завершение: Благодаря бесконечному циклу с вызовом pygame.event.pump() и паузой time.sleep(0.02) внутри функции stop(), после выполнения последней команды главного скрипта окно не зависает. Главный поток продолжает аккуратно рапортовать операционной системе Windows/Linux, что процесс активен, исключая появление системного «колечка ожидания» (статуса «Не отвечает»).
- Выход: Программа закрывается при нажатии на Крестик графического окна или при нажатии кнопки «Стоп» в IDE Olly.

7. Шаблон продвинутого скрипта («Слепой тест» для олимпиад)

Пример программы, где учительский поток динамически выставляет преграды прямо во время заезда, а робот асинхронно сканирует поле дальномером в фоновом потоке, движется по краю линии и останавливается перед внезапной бочкой.

```
from olly.nbase import set_initial, rf, lt, rt, stop, st, KEY,
    IN1, add_block, add_barrel
from olly.bus import dstnc
import _thread

# Глобальная переменная для передачи расстояния между потоками
d = 150

def DThread():
    """Фоновый поток опроса ультразвукового дальномера.
    Блокируется внутри dstnc() на 20 мс, имитируя работу реального датчика.
    """
    global d
    while True:
        d = dstnc()
        st(0.05)

def TeacherTrapThread():
    """Поток преподавателя: подкидывает препятствия прямо во время
    движения"""
    st(3.5) # Робот спокойно едет первые 3.5 секунды...
```

```
    # Внезапно прямо на трассе перед роботом падает оранжевая бочка!
    add_barrel(170, 35)

    st(4.0)
    # Через 4 секунды в другой части поля вырастает стена W
    add_block(120, 80)

# 1. Задаем стартовую точку робота в( сантиметрах)
set_initial(60, 17, 0)


print("Симулятор готов. Переключитесь на окно и ЗАЖМИТЕ SHIFT для старта.")
# 2. Ждем физического нажатия клавиши Shift на ПК
while KEY.value():
    st(0.02)

# 3. Запускаем фоновые потоки
_thread.start_new_thread(DThread, ())
_thread.start_new_thread(TeacherTrapThread, ())

# 4. Главный цикл движения
print("Робот Olly сорвался со старта!")
rf(100)

while True:
    # Если УЗдатчик- зафиксировал преграду ближе 15 см
    # При( упоре вплотную сработает слепая зона и датчик выдаст 150)
    if (d > 0) and (d < 15):
        print(f"Робот[]: Обнаружено препятствие на расстоянии {d} см!
   Экстренное торможение!")
        stop()
        break

    # Релейный регулятор движения по краю сглаженной линии L
    if IN1.read() > 1950:
        rt(1) # Подруливаем вправо
    else:
        lt(1) # Подруливаем влево

    st(0.2) # Пауза для стабильности итераций
```