

## Глава 3

## ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ КОНСТРУКЦИИ

## § 11. Алгоритмы и исполнители

Большинству исполнителей для выполнения алгоритма нужны команды, записанные на формальном языке. Для того чтобы взаимодействие человека и технического устройства было плодотворным, человек обучается формальным языкам, например языкам программирования. Другой путь — совершенствовать исполнителей так, чтобы они понимали естественный язык человека. Человечество пока ещё в начале такого пути, однако уже сегодня существуют исполнители с голосовым управлением.

Первыми бытовыми устройствами с голосовым управлением были стиральные машины и сотовые телефоны. В настоящее время голосовое управление имеют бытовые компьютеры, автомобили, музыкальные центры, кондиционеры, лифты и др.



## 11.1. Понятие алгоритма

Вспомним некоторые понятия, с которыми вы познакомились в 6-м классе.

**Алгоритм** — понятная и конечная последовательность точных действий (команд), формальное выполнение которых позволяет получить решение поставленной задачи.

**Исполнитель алгоритма** — человек, группа людей или техническое устройство, которые понимают команды алгоритма и умеют правильно их выполнять.

**Система команд исполнителя** — команды, которые понимает и может выполнить исполнитель.

Любой исполнитель имеет ограниченную систему команд. Все они разделяются на группы:

- 1) команды, которые непосредственно выполняет исполнитель;
- 2) команды, меняющие порядок выполнения других команд исполнителя.

Компьютер является универсальным исполнителем.

Запись алгоритма в виде последовательности команд, которую может выполнить компьютер, называют **программой**.

Выделяют следующие способы представления алгоритмов (пример 11.1):

- **словесный** — описание алгоритма средствами естественного языка с точной и конкретной формулировкой фраз;

- **графический (блок-схема)** — графическое изображение команд алгоритма с использованием блоков в виде геометрических фигур, отрезков или стрелок, соединяющих эти блоки и указывающих на порядок выполнения команд;

- **программный** — запись алгоритма в виде программы.

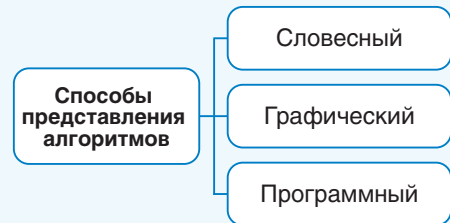
На практике могут применяться комбинации вышеперечисленных способов записи алгоритмов, а также другие способы представления: **табличный**, описание алгоритма с помощью **математических формул**, **псевдокод** (описание структуры алгоритма на частично формализованном языке).

Голосовые помощники Google Assistant и Алиса, созданная компанией Яндекс, могут осуществлять поиск, поддерживать беседу, играть в игры, помогают управлять системой «Умный дом» и многое другое.

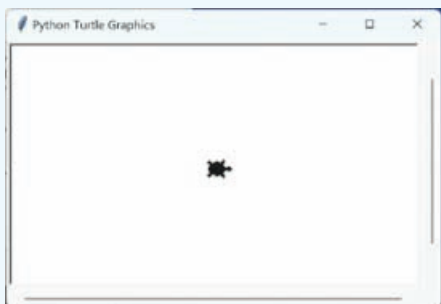


**Пример 11.1.** Способы представления алгоритмов.

На разных этапах решения задачи могут применяться разные способы описания алгоритма. На этапе обсуждения используются словесные и формульные способы. На этапе проектирования рекомендуется использовать графическое представление. Для проверки результатов возможно табличное представление. Программный способ используют на этапе непосредственного применения для решения прикладных задач.



**Пример 11.2.** Среда обитания исполнителя *Черепаша*.



**Пример 11.3.** Основные команды исполнителя *Черепаша*.

| Команда                  | Действие                          |
|--------------------------|-----------------------------------|
| <code>penup()</code>     | Не оставлять след при движении    |
| <code>pendown()</code>   | Оставлять след при движении       |
| <code>forward(X)</code>  | Пройти вперёд X пикселей          |
| <code>backward(X)</code> | Пройти назад X пикселей           |
| <code>left(X)</code>     | Повернуться налево на X градусов  |
| <code>right(X)</code>    | Повернуться направо на X градусов |

## 11.2. Исполнитель *Черепаша*

В прошлом году вы познакомились с исполнителем *Черепаша*, который умеет строить рисунки и чертежи на координатной плоскости.

Среда обитания исполнителя *Черепаша* — координатная плоскость (пример 11.2). Исходное положение *Черепаша* — точка с координатами  $(0, 0)$ , перо опущено.

Основные команды исполнителя *Черепаша* представлены в примере 11.3.

**Пример 11.4.** Прямоугольный участок, длина которого в 2 раза больше ширины, огородили забором длиной 120 метров. Определите длину и ширину участка. Составьте алгоритм и напишите программу, выполнив которую исполнитель *Черепаша* построит чертёж забора этого участка. Масштаб: 1 м соответствует 10 пикселям.

Словесное описание алгоритма вычисления длины и ширины участка:

1. Длина участка в два раза больше ширины, поэтому в сумме длина и ширина составят три одинаковых части. Забор огораживает участок по периметру. Периметр прямоугольника равен

удвоенной сумме длины и ширины, следовательно, он равен шести одинаковым частям.

2. Значение ширины участка получается делением длины забора на шесть частей.

3. Для нахождения значения длины участка удваиваем значение ширины.

Математическая запись действий:

$$1) (1 + 2) \cdot 2 = 6 \text{ (частей).}$$

$$2) 120 : 6 = 20 \text{ (м).}$$

$$3) 20 \cdot 2 = 40 \text{ (м).}$$

Словесное описание алгоритма рисования участка в масштабе 1 : 10:

Установить размеры поля 500 × 300

Поднять перо

В точку (-200, 50)

Опустить перо

Вперёд (400)

Направо (90)

Вперёд (200)

Направо (90)

Вперёд (400)

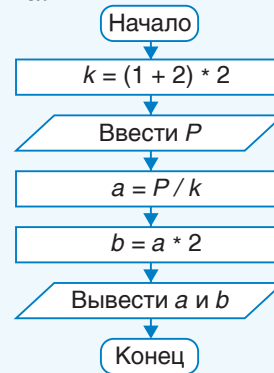
Направо (90)

Вперёд (200)

### 11.3. Алгоритмическая конструкция следование

Существует большое количество алгоритмов, в которых все команды выполняются последовательно одна за другой в том

**Пример 11.4.** Блок-схема алгоритма.



Программа для исполнителя  
*Черепаха:*

```

import turtle
turtle.shape('turtle')
turtle.setup(500, 300)
turtle.penup()
turtle.setpos(-200, 50)
turtle.pendown()

turtle.forward(400)
turtle.right(90)
turtle.forward(200)
turtle.right(90)
turtle.forward(400)
turtle.right(90)
turtle.forward(200)
  
```

Результат работы программы:



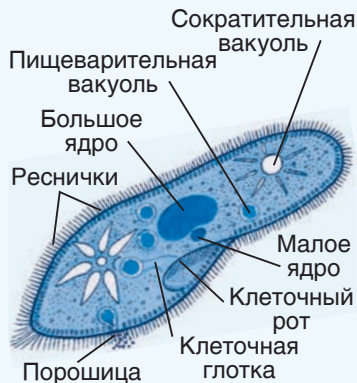
**Пример 11.5.** Алгоритм приготовления бутерброда.

1. Отрезать ломтик батона.
2. Положить на батон лист салата.
3. Отрезать кусочек копчёного мяса.
4. Положить мясо на лист салата.
5. Отрезать кусочек помидора.
6. Положить помидор на мясо.



**Пример 11.6.** Алгоритм выполнения лабораторной работы по биологии «Строение инфузории туфельки».

1. Рассмотреть под микроскопом внешний вид и внутреннее строение инфузории туфельки.
2. Зарисовать инфузорию туфельку и обозначить названия её органоидов.
3. Подвести итог работы.



порядке, в котором они записаны. В таких алгоритмах отсутствуют команды, меняющие порядок выполнения других команд. Программы такого вида вы составляли в прошлом году для исполнителя *Черепашка*.

**Алгоритмическая конструкция следование** — последовательность команд алгоритма, которые выполняются в том порядке, в котором они записаны.

Алгоритмическая конструкция *следование* отображает естественный, последовательный порядок выполнения действий в алгоритме.

*Следование* использовалось в примере 11.4 на с. 73, в котором описывался алгоритм вычисления длины и ширины участка и построения прямоугольника исполнителем *Черепашка*.

Алгоритмическая конструкция *следование* представлена в примерах 11.5 и 11.6.

### 11.4. Вспомогательные алгоритмы

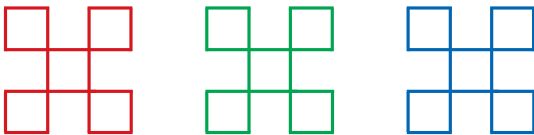
Часто в одной программе нужно рисовать одно и то же изображение несколько раз. Получе-

ние такого изображения удобно оформлять в виде вспомогательного алгоритма, который можно использовать нужное число раз.

**Вспомогательный алгоритм** — это алгоритм, целиком используемый в составе другого алгоритма.

*Вспомогательный алгоритм* решает некоторую подзадачу основной задачи. Вызов вспомогательного алгоритма в программе заменяет несколько команд одной. В языке Python *вспомогательные алгоритмы* записываются в виде функций. Формат описания функции приведён в примере 11.7.

**Пример 11.8.** Написать программу, выполнив которую исполнитель *Черепаха* нарисует следующее изображение:



Данный рисунок состоит из трёх одинаковых фигур. Для рисования одной из них можно оформить вспомогательный алгоритм:

**Пример 11.7.** Формат описания функции.

```
def имя_функции():
    команда1
    команда2
```

Заголовок функции

Команды (тело функции)

**Пример 11.8.** Программа для исполнителя *Черепаха*.

```
import turtle

turtle.shape('turtle')
turtle.setup(670, 240)

def chast():
    turtle.forward(150)
    turtle.right(90)
    turtle.forward(50)
    turtle.right(90)
    turtle.forward(50)
    turtle.right(90)

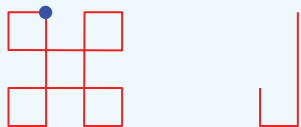
def p(x, y):
    turtle.penup()
    turtle.setpos(x, y)
    turtle.pendown()
    turtle.setheading(270)

def figura(x, y, c):
    p(x, y)
    turtle.color(c)
    chast()
    chast()
    chast()
    chast()

figura(-200, 100, 'red')
figura(0, 100, 'green')
figura(200, 100, 'blue')
```

**Пример 11.8. Продолжение.**

Начальная точка  
и часть фигуры:



Результат выполнения  
программы:



Над проектом по созданию программы может работать несколько человек (или десятков человек). Каждый из членов коллектива делает часть своей работы и оформляет её как отдельный вспомогательный алгоритм.

функцию `figura(x, y, c)`. Параметры  $x$ ,  $y$  и  $c$  определяют координаты точки начала рисования и цвет фигуры. Каждая фигура в свою очередь состоит из четырёх одинаковых частей. Рисование одной части выполняет функция `chast()`. Для перемещения *Черепашки* в позицию с координатами  $(x, y)$  и поворота на  $270^\circ$  используется вспомогательный алгоритм `p(x, y)`.

Описание основного алгоритма:

Рисование фигуры

`(-200, 100, 'red')`

Рисование фигуры

`(0, 100, 'green')`

Рисование фигуры

`(200, 100, 'blue')`



1. Что такое алгоритм?
2. Какие способы записи алгоритмов вам известны?
3. На каком этапе решения задачи используется словесный способ описания алгоритма?
4. Что называют алгоритмической конструкцией *следование*?
5. Какие алгоритмы называются вспомогательными?
6. Для чего нужны вспомогательные алгоритмы?
7. Как описывается вспомогательный алгоритм на языке Python?

**Упражнения**

- 1 Определите, какие рисунки получатся после выполнения *Черепашкой* следующих программ. Изобразите рисунки в тетради и проверьте правильность своих действий, выполнив программы на компьютере.

```

a) import turtle
turtle.shape('turtle')
turtle.setup(500, 500)
turtle.penup()
turtle.setpos(-100, -100)
turtle.pendown()
turtle.setheading(108)
turtle.forward(200)
turtle.right(72)
turtle.forward(200)
turtle.right(72)
turtle.forward(200)
turtle.right(72)
turtle.forward(200)
turtle.right(72)
turtle.forward(200)
turtle.right(72)

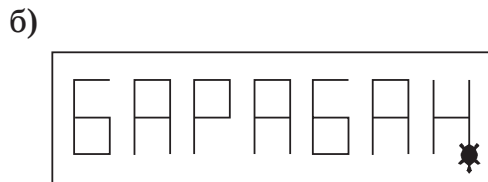
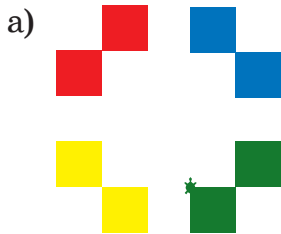
```

```

б) import turtle
turtle.shape('turtle')
turtle.setup(500, 500)
def chast():
    turtle.forward(200)
    turtle.right(90)
    turtle.forward(50)
    turtle.left(90)
    turtle.forward(50)
    turtle.right(90)
turtle.penup()
turtle.setpos(-100, 150)
turtle.pendown()
chast()
chast()
chast()
chast()

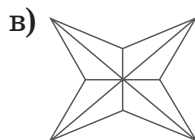
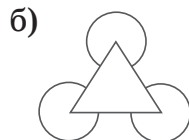
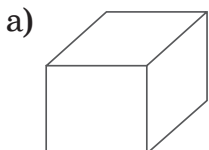
```

3 Напишите для исполнителя *Черепаха* программы получения следующих изображений.



3 Придумайте свои рисунки и составьте программы для их рисования с помощью исполнителя *Черепаха*.

4\* Исполнитель *Чертёжник* может рисовать только отрезки. Проанализируйте рисунки. Какие из них сможет выполнить исполнитель *Чертёжник*? Почему? Все ли из этих рисунков сможет нарисовать исполнитель *Черепаха*?



## § 12. Исполнитель *Робот*

### Роботы на службе человека



Робот на базе трактора  
BELARUS-132



Автоматизированный завод  
по производству комплексов  
с искусственным интеллектом  
холдинга «Горизонт»



Робот-пони для детей

### 12.1. Роботы в жизни человека

Человек с древности мечтал о создании, которое могло бы беспрекословно выполнять его приказы. Сегодня мечта стала реальностью — в жизни человека появились роботы. Они способны выполнять практически любую работу, доступную человеку, а также делать многие вещи, которые ему выполнить сложно или невозможно. Роботы используются на производстве и в быту, могут работать в сфере услуг и развлекать человека. Есть роботы, похожие на человека и совсем непохожие.

**Робот** — автоматическое устройство, которое действует по заранее составленной программе.

Робот получает информацию о внешнем мире от датчиков (аналогов органов чувств живых организмов) и предназначен для осуществления различных операций.

Мир роботов очень разнообразен. В быту современного чело-

века используются автоматические стиральные и посудомоечные машины, роботы-пылесосы. С помощью роботов можно выращивать растения или управлять домом.

На производстве роботы способны выполнять сложные и трудоёмкие работы. Широкое применение роботы находят в транспортировке грузов, расчистке завалов. Роботы эффективны для выполнения опасных и вредных видов работ, таких как опрыскивание полей ядохимикатами, тушение пожаров и исследование опасных объектов.

Постепенно робототехника становится частью образовательного процесса. В Республике Беларусь на базе учреждений общего среднего образования открыты инженерно-технические центры, оснащённые оборудованием для изучения робототехники.

Робот может быть материальным или виртуальным. Виртуальный робот — специальная программа, выполняющая определённые действия. Роботы являются



Российский лунный робот-кентавр<sup>1</sup> FEDOR



Робот-официант в гродненском кафе



Роботизированная рука белорусского производителя манипуляторов Rozum Robotics играет в шашки

<sup>1</sup>Материалы о роботах взяты с сайтов <https://sputnik.by>, <https://iz.ru>, <http://www.robotics.by>, <https://tech.onliner.by>, <http://www.robogeek.ru>, <http://robotrends.ru>.



Робототехническое оборудование для инженерно-технических центров



Программируемые роботы (конструктор LEGO Education)

В отличие от исполнителя *Черепашка*, исполнитель *Робот* не входит в набор стандартных модулей языка Python. Сегодня можно выбрать несколько разных реализаций *Робота* на этом языке. В учебном пособии использован *Робот*, разработанный МФТИ<sup>1</sup>.

Разработанная для исполнителя *Робот* библиотека была незначительно изменена и дополнена оригинальным задачиком, в который входят все примеры и упражнения из учебного пособия.

исполнителями. Виртуального робота можно рассматривать как компьютерного исполнителя. Для исполнителей обычно определяют среду обитания и систему команд.

Общим для всех роботов является то, что человек может ими управлять. Робот может получать команды от оператора и выполнять их по одной или действовать автономно по предварительно составленной программе.

## 12.2. Среда обитания и система команд компьютерного исполнителя *Робот*

Средой обитания исполнителя *Робот* является прямоугольное клетчатое поле. Между некоторыми клетками, а также по периметру поля находятся стены. Серое кольцо обозначает начальное положение *Робота*, чёрная точка — конечное. *Робот* может передвигаться по полю и закрашивать указанные клетки. В заданиях из встроенного задачника *Робота* закрашенные клетки помечены серо-голубым цветом, те клетки, которые *Робот* дол-

<sup>1</sup> [http://judge.mipt.ru/mipt\\_cs\\_on\\_python3/labs/lab2.html](http://judge.mipt.ru/mipt_cs_on_python3/labs/lab2.html).

жен закрасить, — светло-жёлтым (пример 12.1).

Поле *Робота*, на котором определено положение стен, начальное и конечное положение исполнителя, а также клетки, которые уже закрасены и (или) должны быть закрасены, называют **обстановкой**.

Для подключения исполнителя *Робот* в программе прописывается команда `from pyrob.api import*`. Готовые задания с обстановками для *Робота* хранятся в задачнике и вызываются командой `@task`.

Структура программы для записи решения задачи из задачника показана в примере 12.2. Имя функции в программе должно соответствовать имени задачи. Все имена задач, которые соответствуют примерам из учебного пособия, начинаются со слова `prim`. Задачи, соответствующие упражнениям, начинаются с `urp`. Далее, после слов `prim` или `urp` следует соответствующий номер. В теле функции записываются команды исполнителя *Робот*.

**Пример 12.1.** Поле исполнителя *Робот* с начальной обстановкой.



**Пример 12.2.** Вызов задачи `prim_12_1` из встроенного задачника.

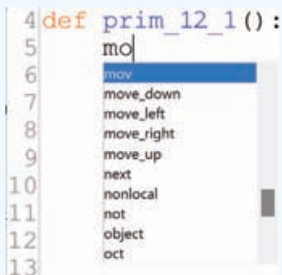
```

*12_1.py - C:/prog_7ki/robot/9_1.p...
File Edit Format Run Options Window Help
1 from pyrob.api import *
2
3 @task
4 def prim_12_1():
5
6
7 run_tasks()
8
Ln: 8 Col: 0

```

Библиотека `pyrob` с задачником и примерами из учебного пособия размещена на образовательном портале <https://eior.by>. После скачивания и распаковки архива свои программы можно сохранять в ту же папку `robot`, в которой сохранены примеры. В этом случае установка библиотеки `pyrob` с помощью менеджера пакетов `pip` не потребуется.

**Пример 12.3.** Выбор команды для *Робота* из выпадающего списка.



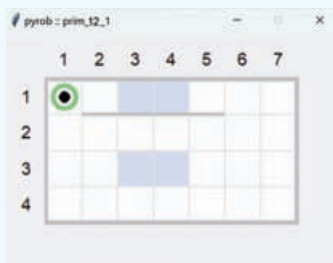
**Пример 12.4.** Решение задачи `prim_12_1`.

```
from pyrob.api import *

@task
def prim_12_1():
    move_right(4)
    move_up()
    move_left(2)
    fill_cell()
    move_left()
    fill_cell()
    move_left(2)

run_tasks()
```

**Пример 12.5.** Обстановка *Робота* после выполнения программы.



Система команд исполнителя *Робот*.

| Команда                   | Действие                        |
|---------------------------|---------------------------------|
| <code>move_right()</code> | Перемещает <i>Робота</i> вправо |
| <code>move_left()</code>  | Перемещает <i>Робота</i> влево  |
| <code>move_up()</code>    | Перемещает <i>Робота</i> вверх  |
| <code>move_down()</code>  | Перемещает <i>Робота</i> вниз   |
| <code>fill_cell()</code>  | Закрашивает текущую ячейку      |

Команды при наборе можно выбирать из выпадающего списка. Для этого достаточно набрать первые буквы команды и нажать `Ctrl + пробел` (пример 12.3). Команды перемещения *Робота* могут содержать в качестве параметра число, которое указывает, на сколько клеток должен переместиться *Робот* в указанном направлении. Если число не указано, то *Робот* сдвигается на одну клетку.

Решение задачи `prim_12_1` приведено в примере 12.4. Если задача решена верно, кольцо, изображающее *Робота*, окрашивается в зелёный цвет (пример 12.5).

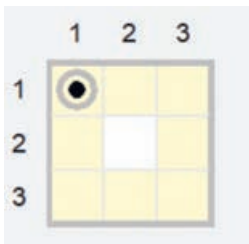
*Робот* может становиться на любую клетку. *Робот* не может

переместиться с клетки на клетку, если они разделены стеной. *Робот* не может переместиться за границы поля. При попытке выполнить недопустимое действие *Робот* изображается красным кругом (пример 12.6). *Робот* может закрасить уже закрашенную клетку, такое действие ошибку не вызывает. Если *Робот* выполнил все команды, но закрасил не все клетки, или закрасил лишние клетки, или оказался не в конечной точке, то он изображается серым кругом (пример 12.7).

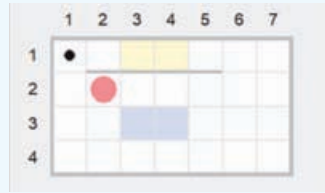
### 12.3. Использование алгоритмической конструкции следование для исполнителя *Робот*

Рассмотрим примеры решения задач для исполнителя *Робот*.

**Пример 12.8.** *Робот* находится на поле размером  $3 \times 3$  клетки. Нужно закрасить все клетки, кроме средней.

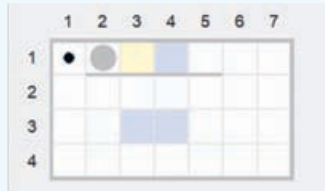


**Пример 12.6.** *Робот* врезался в стену.



Эта ситуация возникнет, если первой командой *Робота* будет команда `move_up()`.

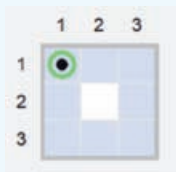
**Пример 12.7.** *Робот* не закрасил одну клетку и не пришёл в конечную точку.



**Пример 12.8.** Программа для *Робота*.

```
from pyrob.api import *
@task
def prim_12_8():
    fill_cell()
    move_right()
    fill_cell()
    move_right()
    fill_cell()
    move_down()
    fill_cell()
    move_down()
    fill_cell()
    move_left()
    fill_cell()
    move_left()
    fill_cell()
    move_up()
    fill_cell()
    move_up()
run_tasks()
```

**Пример 12.8. Продолжение.**  
Результат работы программы:



Другой способ записи программы для *Робота*:

```
from pyrob.api import *
@task
def prim_12_8():
    fill_cell(), move_right()
    fill_cell(), move_right()
    fill_cell(), move_down()
    fill_cell(), move_down()
    fill_cell(), move_left()
    fill_cell(), move_left()
    fill_cell(), move_up()
    fill_cell(), move_up()
run_tasks()
```

**Пример 12.9.** Программа для *Робота*.

```
from pyrob.api import *
@task
def prim_12_9():
    move_down()
    fill_cell()
    move_right(), move_up()
    fill_cell()
    move_right(), move_up()
    fill_cell()
    move_left(4)
    fill_cell()
    move_right()
    move_down()
    fill_cell()
    move_right()
run_tasks()
```

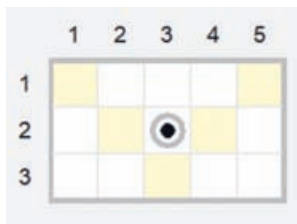
Для решения задачи *Робот* должен выполнить следующий алгоритм:

Закрасить, вправо  
Закрасить, вправо  
Закрасить, вниз  
Закрасить, вниз  
Закрасить, влево  
Закрасить, влево  
Закрасить, вверх  
Закрасить, вверх

В данном алгоритме *Робот* обходит клетки, двигаясь по часовой стрелке. Тот же результат можно было бы получить, если бы *Робот* обходил поле против часовой стрелки, изначально двигаясь вниз.

При записи команд в программе в одной строке можно располагать несколько команд. В этом случае они отделяются запятой или точкой с запятой. Для исполнителя *Робот* такой способ записи делает программу более компактной и, иногда, более понятной.

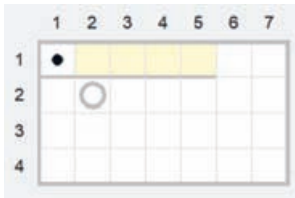
**Пример 12.9.** Составить программу для закраски клеток поля *Робота* по образцу.



Для решения задачи *Робот* должен выполнить следующий алгоритм:

Вниз  
 Закрасить  
 Вправо, вверх  
 Закрасить  
 Вправо, вверх  
 Закрасить  
 Влево на 4 клетки  
 Закрасить  
 Вправо  
 Вниз  
 Закрасить  
 Вправо

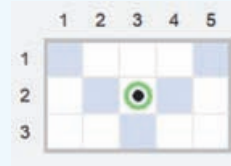
**Пример 12.10.** Составить программу для закраски клеток поля *Робота* по образцу.



Для решения задачи *Робот* должен обойти линию (стену), закрасить указанные клетки, переместиться в клетку, определяющую конечное положение. Начальная обстановка похожа на обстановку из примера 12.1 на с. 81, поэтому можно воспользоваться уже имеющейся программой (пример 12.4 на с. 82) и внести в неё нужные изменения.

**Пример 12.9.** *Продолжение.*

Результат работы программы:



**Пример 12.10.** Изменение программы для *Робота* из примера 12.4.

```
from pyrob.api import *
@task
def prim_12_10():
    move_right(4)
    move_up()
    move_left(2)
    fill_cell()
    move_left()
    fill_cell()
    move_left(2)
run_tasks()
```

10

Скопировать и вставить 2 раза перед последней командой

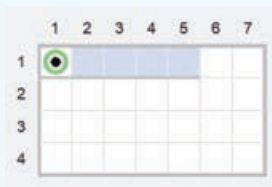
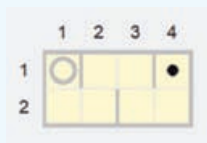
Для более компактной записи можно объединить команды `move_left()`, `fill_cell()` в одну строку.

Изменённая программа:

```
from pyrob.api import *
@task
def prim_12_10():
    move_right(4)
    move_up()
    move_left(), fill_cell()
    move_left(), fill_cell()
    move_left(), fill_cell()
    move_left(), fill_cell()
    move_left()
run_tasks()
```

**Пример 12.10. Продолжение.**

Результат работы программы:

**Пример 12.11. Начальная обстановка.**

Программа 1  
для исполнителя *Робот*:

```
from pyrob.api import *

@task
def prim_12_11():
    fill_cell(), move_down()
    fill_cell(), move_right()
    fill_cell(), move_up()
    fill_cell()
    move_right()
    fill_cell(), move_down()
    fill_cell(), move_right()
    fill_cell(), move_up()
    fill_cell()

run_tasks()
```

Результат работы программы:



Алгоритм решения задачи:

Вправо на 4 клетки  
Вверх  
Влево, закрасить  
Влево, закрасить  
Влево, закрасить  
Влево, закрасить  
Влево

В примерах 12.8–12.10 команды исполнителя *Робот* выполнялись последовательно, одна за другой, в том порядке, в котором они были записаны. Все приведённые алгоритмы реализованы с использованием алгоритмической конструкции *следование*.

**12.4. Вспомогательные алгоритмы для исполнителя *Робот***

**Пример 12.11.** *Робот* должен закрасить все клетки на поле размером  $2 \times 4$ . Но двигаться по прямой линии ему мешают стены, которые нужно обойти.

Алгоритм решения задачи:

Закрасить, вниз  
Закрасить, вправо  
Закрасить, вверх  
Закрасить  
Вправо  
Закрасить, вниз  
Закрасить, вправо  
Закрасить, вверх  
Закрасить

Если проанализировать алгоритм, то можно заметить, что дважды повторяется последовательность команд, которая закрашивает квадрат из четырёх клеток:

```
Закрасить, вниз
Закрасить, вправо
Закрасить, вверх
Закрасить
```

Оформим эти команды как вспомогательный алгоритм, который назовём квадрат. Тогда алгоритм решения исходной задачи может быть записан так:

```
Квадрат
Вправо
Квадрат
```

Использование вспомогательного алгоритма при решении этой задачи позволило не записывать дважды одну и ту же последовательность команд. Вспомогательные алгоритмы можно использовать и в том случае, когда исходная задача разбивается на несколько независимых друг от друга задач. Тогда каждую из них можно оформить как вспомогательный алгоритм.

### Пример 12.11. Продолжение.

Программа 2 (с использованием вспомогательного алгоритма) для исполнителя *Робот*.

```
from pyrob.api import *

@task
def prim_12_11():
    def kvadrat():
        fill_cell(), move_down()
        fill_cell(), move_right()
        fill_cell(), move_up()
        fill_cell()

    kvadrat()
    move_right()
    kvadrat()

run_tasks()
```

При наборе текста программы обратите внимание на расположение описания вспомогательного алгоритма `kvadrat()` в программе и правильную расстановку отступов при записи команд. Функция `kvadrat()` располагается внутри функции `prim_12_11`, поэтому для всех команд, составляющих тело функции `kvadrat()`, необходим двойной отступ.

При неправильном отступе получим ошибку.



**Пример 12.12.** Программа для исполнителя *Робот*.

```

from pyrob.api import *

@task
def prim_12_12():
    def krest():
        move_left()
        fill_cell(), move_left()
        fill_cell(), move_left()
        fill_cell()
        move_down(), move_right()
        fill_cell(), move_up()
        fill_cell(), move_up()
        fill_cell()

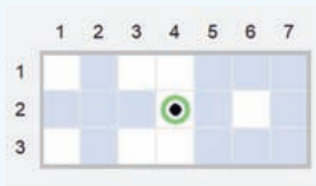
    def ramka():
        fill_cell(), move_right()
        fill_cell(), move_right()
        fill_cell(), move_down()
        fill_cell(), move_down()
        fill_cell(), move_left()
        fill_cell(), move_left()
        fill_cell(), move_up()
        fill_cell(), move_up()

    krest()
    move_right(3)
    ramka()
    move_left()
    move_down()

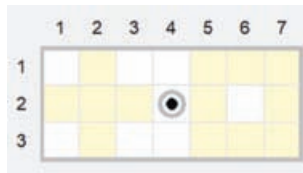
run_tasks()

```

Результат работы программы:



**Пример 12.12.** Написать программу для закраски клеток поля *Робота* по образцу.



В данной задаче *Робот* должен нарисовать две отдельные фигуры: крест и рамку. Составим два вспомогательных алгоритма: крест и рамка.

Вспомогательный алгоритм крест:

```

Влево
Закрасить, влево
Закрасить, влево
Закрасить, вниз, вправо
Закрасить, вверх
Закрасить, вверх
Закрасить

```

В качестве вспомогательного алгоритма рамка можно использовать программу из примера 12.8 на с. 83. Для перехода от одной фигуры к другой нужно сдвинуться на 3 клетки вправо, а после рисования рамки — вернуться в исходную клетку. Описание основного алгоритма:

```

Крест
Вправо на 3 клетки
Рамка
Влево, вниз

```

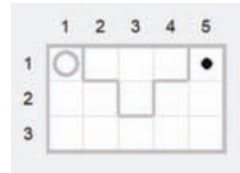


1. Что такое робот?
2. Какие команды входят в систему команд учебного компьютерного исполнителя *Робот*?
3. Опишите среду обитания учебного исполнителя *Робот*.
4. В чём особенность применения алгоритмической конструкции *следование*?
5. В каких случаях полезно использовать вспомогательные алгоритмы?

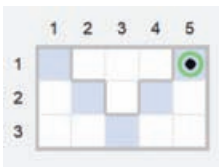


## Упражнения

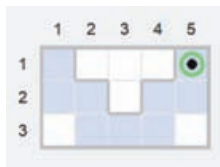
1 Начальная обстановка на поле *Робота* изображена на рисунке справа. Трое учащихся составили и выполнили алгоритм, по которому *Робот* закрасил все клетки пути от начальной к конечной. На каком из рисунков — а, б или в — изображено решение этой задачи? Поясните.



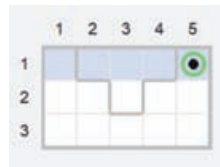
а)



б)



в)



2 Определите, какой из приведённых алгоритмов решает задачу, сформулированную в упражнении 1. Объясните, почему другие программы не могут быть алгоритмами решения данной задачи.

а)

```
fill_cell(), move_down()
fill_cell(), move_right()
fill_cell(), move_down()
fill_cell(), move_right()
fill_cell(), move_right()
fill_cell(), move_up()
fill_cell(), move_right()
fill_cell(), move_up()
fill_cell()
```

б)

```
fill_cell()
move_down()
move_right()
fill_cell()
move_down()
move_right()
fill_cell()
move_right()
move_up()
fill_cell()
move_right()
move_up()
fill_cell()
```

в)

```
turtle.right(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
```

3 Определите, для какого исполнителя приведён алгоритм в упражнении 2, пункт в). Сформулируйте для этого исполнителя задачу, решением которой будет приведённый алгоритм.

4 Предложите другие способы решения задачи из примера 12.9.

5 Для исполнителя *Робот* была составлена следующая программа:

```
fill_cell()
move_right()
move_up()
fill_cell()
move_right()
move_down()
```

Изобразите в тетради «узор», который нарисует *Робот*. При каких минимальных размерах поля *Робот* сможет выполнить данную программу?

6 Все команды в программе из упражнения 5 учащийся скопировал три раза. Как изменится «узор» после выполнения программы? Как можно по-другому записать этот алгоритм? Какого размера поле нужно создать? Подсказка: воспользуйтесь вспомогательным алгоритмом.

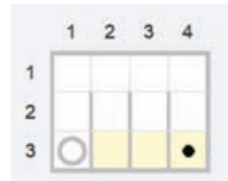
7 Программа решения задачи была записана на доске. Двое учащихся, записывая этот алгоритм для исполнителя *Робот*, пропустили из-за невнимательности по одной команде из этой программы. Какую команду пропустил каждый из учащихся? Что будет результатом работы каждой программы? Проверьте свой ответ на компьютере.

| Программа,<br>записанная первым учащимся  | Программа,<br>записанная вторым учащимся  |
|---|---|
| <pre>from pyrob.api import * @task def upr_12_7():     fill_cell(), move_right()     fill_cell(), move_right()     fill_cell(), move_down()     fill_cell(), move_down()     fill_cell(), move_down()     move_left()     move_down()     move_left()</pre> | <pre>from pyrob.api import * @task def upr_12_7():     fill_cell(), move_right()     fill_cell(), move_right()     fill_cell(), move_down()     fill_cell(), move_down()     fill_cell(), move_down()     move_left()     fill_cell()     move_left()</pre> |

Продолжение таблицы

| Программа,<br>записанная первым учащимся   | Программа,<br>записанная вторым учащимся  |
|--|---|
| <pre> fill_cell(), move_down() fill_cell(), move_down() fill_cell(), move_right() fill_cell(), move_right() fill_cell(), move_up() fill_cell(), move_up() fill_cell(), move_up() move_left() fill_cell(), move_up() move_left() fill_cell(), move_up() fill_cell(), move_up() run_tasks()                     </pre> | <pre> fill_cell(), move_down() fill_cell(), move_down() fill_cell(), move_right() fill_cell(), move_right() fill_cell(), move_up() fill_cell(), move_up() fill_cell(), move_up() fill_cell(), move_up() move_left() fill_cell(), move_up() move_left() fill_cell(), move_up() fill_cell(), move_up() run_tasks()                     </pre> |

**8** Составьте программу для решения задачи `upr_12_8`. Реализуйте два алгоритма: один с использованием алгоритмической конструкции *следование*, другой — с использованием вспомогательного алгоритма. Сравните решения.



## § 13. Алгоритмическая конструкция повторение

### 13.1. Алгоритмы с циклами

В окружающем мире можно наблюдать ситуации, при которых различные действия, процессы и события повторяются. Одни повторяются несколько раз и завершаются. Другие могут повторяться очень долго (например, круговорот воды в природе, бесконечное движение планет в космическом пространстве и т. д.).

Понятие цикла используется в различных сферах человеческой деятельности. Под циклом понимают совокупность явлений, процессов, составляющих кругооборот в течение определённого промежутка времени. С этой точки зрения можно говорить о годовом цикле вращения Земли вокруг Солнца или о производственном цикле.

**Пример 13.1.** Приготовление пельменей.

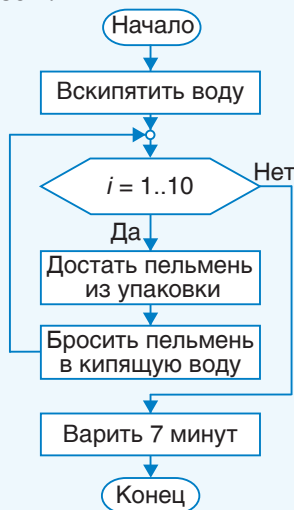


Алгоритм:

1. Вскипятить воду.
2. Для  $i$  от 1 до 10 повторять:
  - а) достать пельмень из упаковки;
  - б) бросить пельмень в кипящую воду.
3. Варить 7 минут.

В данном примере параметр цикла  $i$  изменяется от 1 до 10. Действия «Достать пельмень из упаковки» и «Бросить пельмень в кипящую воду» составляют тело цикла. Тело цикла выполнится 10 раз (для значений  $i = 1, 2, \dots, 10$ ).

Блок-схема данного алгоритма будет выглядеть следующим образом:



Человеку также регулярно приходится выполнять повторяющиеся действия: умываться, готовить еду, посещать парикмахерскую, ходить на работу и др.

Как правило, человек составляет программы, в которых каждая команда в отдельности и весь алгоритм в целом выполняются за конечное число повторений.

**Алгоритмическая конструкция повторение (цикл)** определяет последовательность действий, выполняемых многократно. Эту последовательность действий называют **телом цикла**.

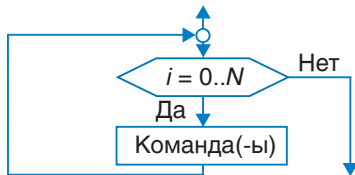
Существует несколько способов управления тем, сколько раз будет повторяться тело цикла.

Команда **цикл с параметром (цикл со счётчиком, цикл «для»)** — способ организации алгоритмической конструкции **повторение (цикл)**, при котором количество повторов зависит от начального и конечного значений параметра цикла.

*Цикл с параметром* организует выполнение команд тела цикла заранее известное число раз (пример 13.1).

Параметр цикла играет роль счётчика и определяет нумерацию повторов тела цикла. Часто нумерацию начинают с 0 и ограничивают числом  $N$ . Цикл выполнится  $N$  раз для значений  $0, 1, \dots, N - 1$ . Если нумерация установлена двумя произвольными числами  $N1$  (начальное значение) и  $N2$  (конечное значение), то цикл выполнится  $(N2 - N1)$  раз (примеры 13.2, 13.3).

Алгоритмическая конструкция цикла с параметром может изображаться на блок-схеме следующим образом:



В этой конструкции в прямоугольнике(-ах) записываются повторяющиеся команды алгоритма (тело цикла), которые выполняются  $N$  раз (да). При этом после каждого выполнения команд тела цикла происходит проверка, сколько раз уже выполнен цикл. На блок-схеме переход на проверку условия изображается в виде стрелки, выходящей из тела цикла и возвращающейся

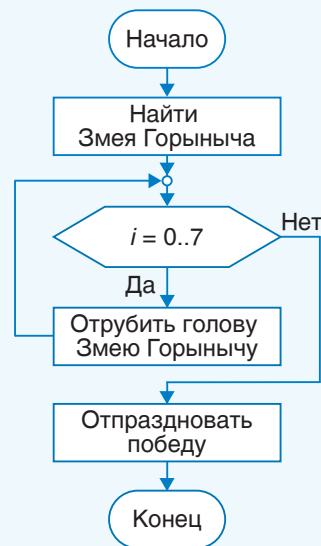
**Пример 13.2.** Вычислить  $a^n$  (например,  $3^5 = 243$ ). Алгоритм возведения числа в степень может быть описан следующим образом:

1. Ввести значения  $a$  и  $n$ .
2. Определить начальное значение результата  $r = 1$ .
3. Для  $i$  от 0 до  $n$  повторять: умножить результат на  $a$ .
4. Записать результат.

**Пример 13.3.** В фольклорных произведениях часто встречается многоголовый Змей Горыныч (количество голов может быть равно, например, 7). Алгоритм победы над Змеем Горынычем может быть описан следующим образом:

1. Найти Змея Горыныча.
2. Для  $i$  до 7 повторять: отрубить голову Змею Горынычу.
3. Отпраздновать победу.

Блок-схема данного алгоритма:



**Пример 13.4.** Работа цикла `for i in range (N1, N2)`.

Вначале цикл выполняется для значения  $i = N1$ . Затем значение  $i$  увеличивается на 1 и цикл выполняется для значения  $i = N1 + 1$ . Так продолжается до тех пор, пока  $i$  не достигнет значения  $N2$ . При значении  $i = N2$  цикл завершает работу и команды в теле цикла больше не выполняются. Последний раз команды в теле цикла выполняются для значения  $i = N2 - 1$ .

Многие роботы, которые используются в быту и на производстве, могут выполнять циклические алгоритмы. Примером такого робота является суши-робот, который может производить от 450 до 4000 заготовок для суши в час.



к проверке. Как только команды тела цикла выполняются  $N$  раз (нет), цикл завершится. Если  $N \leq 0$ , то команды из тела цикла не выполняются ни разу.

Для записи цикла с параметром в языке Python используется команда `for`. Формат записи команды:

```
for i in range (N1, N2):
    тело цикла
```

Строка `for i in range (N1, N2):` является заголовком цикла. Эту строку можно прочитать следующим образом: «Для переменной  $i$  в диапазоне от  $N1$  до  $N2$  делай...». Переменную  $i$  называют **переменной (параметром) цикла**. Одно выполнение тела цикла называют **итерацией цикла**. Процесс работы цикла `for` описан в примере 13.4.

Команды тела цикла записываются со сдвигом относительно заголовка цикла.

В языке Python существуют другие варианты записи цикла `for`. Если нужно указать только количество выполнений цикла, то его можно записать так:

```
for i in range (N):
    тело цикла
```

Такая запись соответствует записи:

```
for i in range (0, N):
    тело цикла
```

Если переменная цикла должна изменяться не на 1, то можно задать шаг её изменения:

```
for i in range (N1, N2, h):
    тело цикла
```

В этом случае при каждой итерации цикла значение переменной  $i$  будет изменяться на величину  $h$ . Значение  $h$  может быть положительным или отрицательным целым числом, но не может равняться нулю (пример 13.5).

### 13.2. Использование команды цикл с параметром для исполнителя Робот

**Пример 13.6.** Написать программу для закрашивания 10 клеток поля *Робота* по образцу.

Для закрашивания всех 10 клеток поля нужно в цикле выполнить 10 раз команды:

```
Закрасить
Вправо
```

Данные команды образуют тело цикла.

Командами, образующими тело цикла, могут быть любые команды из системы команд исполнителя. Кроме того, в теле цикла

**Пример 13.5.** Варианты цикла `for`.

Уменьшение переменной цикла на 1:

```
for i in range (5, 0, -1):
```

Переменная цикла будет принимать значения: 5, 4, 3, 2, 1.

Увеличение переменной цикла на 10:

```
for i in range (-20, 30, 10):
```

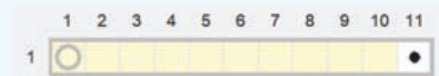
Переменная цикла будет принимать значения: -20, -10, 0, 10, 20.

Использование только нечётных чисел от 1 до 100:

```
for i in range (1, 100, 2):
```

Переменная цикла будет принимать значения: 1, 3, 5, ..., 97, 99.

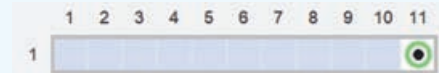
**Пример 13.6.** Начальная обстановка.



Программа  
для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_13_5():
    for i in range (10):
        fill_cell()
        move_right()
run_tasks()
```

Результат работы программы:



**Пример 13.7.** Начальная обстановка.



Программа  
для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_13_7():
    def kvadrat():
        fill_cell()
        move_down()
        fill_cell()
        move_left()
        fill_cell()
        move_up()
        fill_cell()
        move_left()

    for i in range (5):
        kvadrat()

run_tasks()
```

может вызываться вспомогательный алгоритм. Использование вспомогательного алгоритма позволяет сократить запись тела цикла и делает программу более понятной.

**Пример 13.7.** Написать программу для закрашивания клеток поля *Робота* по образцу.

На поле исполнителя *Робот* присутствуют стены. При обходе стен *Робот* выполняет следующие команды:

Закрасить, вниз  
Закрасить, влево  
Закрасить, вверх  
Закрасить, влево

Для решения задачи данную группу команд *Робот* должен повторить 5 раз. Оформим эти команды как вспомогательный алгоритм *kvadrat* и вызовем в цикле этот алгоритм. В данном примере тело цикла состоит из одной команды *kvadrat*.

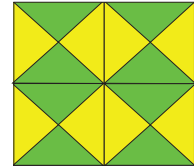


1. Какие циклические процессы, происходящие в окружающем мире, вы можете назвать?
2. Что понимают под алгоритмической конструкцией *повторение*?
3. Что такое тело цикла?
4. Что такое команда *цикл с параметром*?
5. Какие ключевые слова входят в описание заголовка цикла с параметром в Python?
6. Что такое итерация цикла?
7. Какие есть способы записи команды цикла в языке Python?

   **Упражнения**

**1** Опишите словесно или изобразите с помощью блок-схемы следующие алгоритмы.

1. Рисование в графическом редакторе следующего изображения (повторить 4 раза рисование квадрата с диагоналями и закрасненными областями).



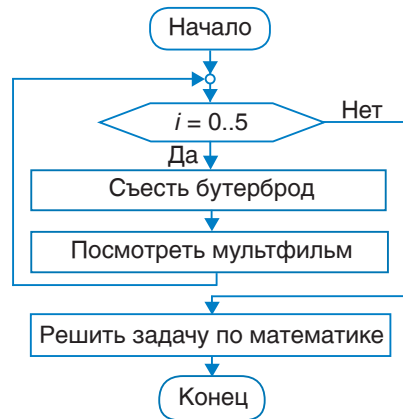
2. Сверление 10 отверстий.

3. Сервировка стола к обеду на 6 персон.

4. Каждую минуту бактерия делится надвое. Изначально есть одна бактерия. За бактериями наблюдали 10 минут. Определите количество бактерий в конце наблюдения. Заполните таблицу согласно алгоритму.

|                     |   |   |   |   |   |   |   |   |   |   |    |
|---------------------|---|---|---|---|---|---|---|---|---|---|----|
| Время, мин          | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Количество бактерий | 1 | 2 |   |   |   |   |   |   |   |   |    |

**2** Семиклассник Вася после школы пригласил своего друга Колю помочь ему в решении 5 задач по математике. В гостях Коля посоветовал Васе провести остаток дня, воспользовавшись следующим алгоритмом, записанным в виде блок-схемы. Почему Вася получил двойку по математике?



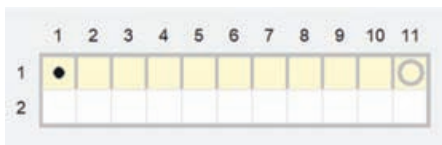
**3** Составьте программу для решения задачи `upr_13_3`. Сравните алгоритм решения этой задачи с примером 13.5. Что у них общего? Чем они отличаются?



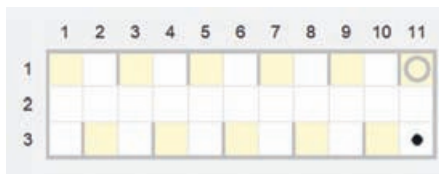
4 Составьте программу для решения задачи `upr_13_4`. Сравните её решение с программой из упражнения 3 и примера 13.5.



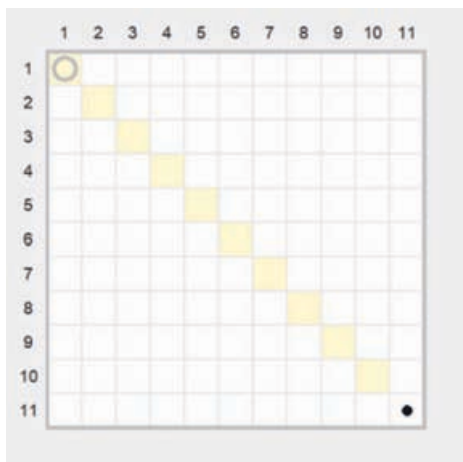
5 Составьте программу для решения задачи `upr_13_5`. Используйте вспомогательный алгоритм для решения задачи.



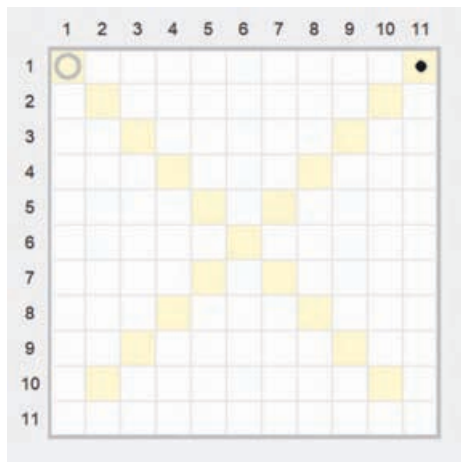
6\* Составьте программу для решения задачи `upr_13_6`. Используйте вспомогательный алгоритм для решения задачи.



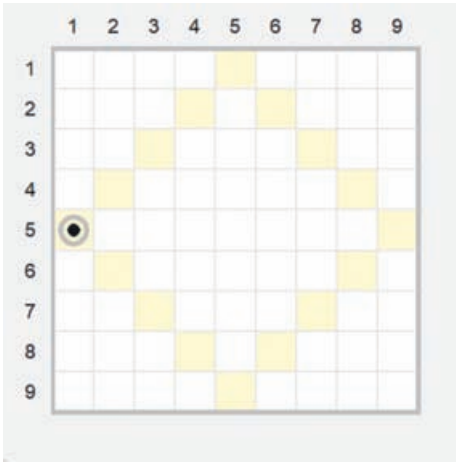
7 Составьте программу для решения задачи `upr_13_7`.



8\* Составьте программу для решения задачи `upr_13_8`.



9 Для решения задачи `upr_13_9` Петя составил алгоритм и записал программу. Петин младший брат Олег удалил несколько команд. Сколько команд удалил Олег? Восстановите программу, которую написал Петя.



```

from pyrob. api import *
@task
def upr_13_9():
    for i in range (4):
        fill_cell()
        move_right()
        move_down()
    for i in range (4):
        move_right()
        move_up()
    for i in range (4):
        fill_cell()
        move_down()

run_tasks()

```

**10** Вася пытается представить, как можно было бы использовать роботов в различных ситуациях, описанных в литературных произведениях. Например, для Тома Сойера, которого тётушка Поли отправила красить забор, Вася придумал робота-маляра. Вася решил, что такому роботу в системе команд достаточно одной команды: покрась доску. Алгоритм покраски забора из 20 досок Вася записал так:

1. Установить робота у левого края забора
2. Для  $i = 0..20$  повторять
3.     Покрась доску

Сможет ли робот-маляр покрасить забор? В чём ошибка Васи? Исправьте алгоритм, добавив необходимую(-ые) команду(-ы).

## § 14. Использование условий

### 14.1. Условия для исполнителя

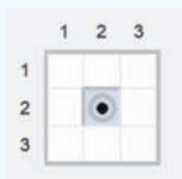
#### *Робот*

Как и многие другие исполнители, исполнитель *Робот* может не только выполнять действия, но и проверять условия.

Условием для *Робота*, как и для любого исполнителя, является

Робот-пылесос проверяет истинность условий с помощью датчиков. Датчики расстояния позволяют ему «видеть» препятствия и не касаться их. Если он наезжает на препятствие под углом, то срабатывают датчики столкновения. В этом случае пылесос меняет своё направление по заданному алгоритму.

**Пример 14.1.** Начальная обстановка *Робота*.



В данном случае для *Робота* будут истинны следующие условия:

```
wall_up()
wall_left()
cell_is_filled()
```

Ложными будут условия:

```
wall_down()
wall_right()
```

**Пример 14.2.** Сбор грибов.

Использование цикла с параметром при составлении алгоритма решения этой задачи может привести к разным результатам. Или корзина может быть полупустой, или не все найденные грибы в неё поместятся.



понятное ему высказывание, которое может быть истинным (соблюдаться) либо ложным (не соблюдаться). Рассмотрим систему условий для компьютерного исполнителя *Робот*.

| Условие          | Результат                              |
|------------------|--|
| wall_up()        | Истинно, если стена <i>сверху</i>      |
| wall_down()      | Истинно, если стена <i>снизу</i>       |
| wall_left()      | Истинно, если стена <i>слева</i>       |
| wall_right()     | Истинно, если стена <i>справа</i>      |
| cell_is_filled() | Истинно, если текущая клетка закрашена |

Истинность условий для исполнителя *Робот* проверяется для конкретной обстановки (пример 14.1).

## 14.2. Команда *цикл с предусловием*

Цикл с параметром используется при составлении алгоритма в том случае, когда заранее известно количество повторений. Однако часто до выполнения цикла количество повторений неизвестно.

**Пример 14.2.** Вы с родителями пошли в лес собирать грибы (предполагается, что они там есть). Ваши действия можно описать командами: найти гриб, срезать гриб, положить гриб в корзину. Эти действия будут выполняться в цикле, но вы заранее не знаете, сколько грибов поместится в корзину. Поэтому можно говорить не о количестве повторений (количество грибов), а об условии, при котором вы будете продолжать сбор грибов: пока корзина не заполнена.

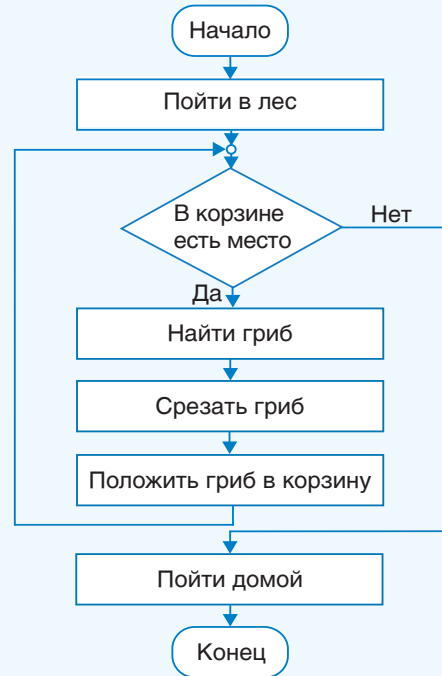
**Команда цикл с условием (цикл «пока»)** — это такой способ организации алгоритмической конструкции *повторение* (цикл), при котором количество выполнений команд тела цикла зависит от истинности или ложности условия цикла.

*Цикл с условием* используется в том случае, когда количество повторений тела цикла заранее неизвестно, но известно условие продолжения работы.

Условие цикла определяет, как долго будет выполняться цикл.

**Пример 14.2. Продолжение.**

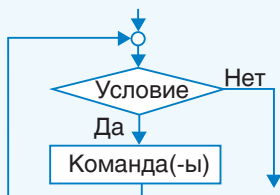
При использовании цикла с условием домой унесём полную корзину грибов.



Циклы выполняются до тех пор, пока условие цикла истинно. Иногда нужно прервать выполнение цикла без проверки условия. Для этого используют команду `break`, которая прекращает выполнение цикла и передаёт управление команде, следующей за циклом.

Иногда возникает необходимость принудительно начать следующую итерацию цикла, пропустив часть команд в его теле. Для таких случаев существует ключевое слово `continue`.

**Пример 14.3.** Блок-схема цикла с предусловием.



**Пример 14.4.** Принцип работы команды *цикл с предусловием*.

В качестве условия записывается логическое выражение, которое определяет, как долго будет выполняться цикл. В теле цикла записываются повторяющиеся команды алгоритма, которые выполняются пока верно условие цикла (да).

На блок-схеме команды тела цикла заключены в прямоугольники, а условие — в ромб.

Сначала проверяется условие, и если оно истинно, то выполняются команды тела цикла. После каждого выполнения команд тела цикла автоматически происходит возврат к проверке истинности условия.

Как только условие станет ложным (нет), цикл завершится.

Количество итераций цикла будет на 1 меньше, чем количество проверок истинности условия цикла. Если условие сразу ложно, то цикл не выполнится ни разу, т. е. количество итераций цикла будет равно нулю.

Пока условие истинно, выполняются команды, составляющие тело цикла. Цикл прекращает выполняться тогда, когда условие становится ложным. *Цикл с предусловием* имеет такое название, поскольку проверка условия предваряет выполнение команд тела цикла.

В примере 14.3 показана блок-схема *цикла с предусловием*. Принцип работы *цикла с предусловием* описан в примере 14.4.

Если условие в цикле будет всегда истинно (всегда да), то такой цикл не сможет завершиться. Возникшую ситуацию называют **зацикливанием**.

Для записи цикла с предусловием в Python используется команда `while`. Формат записи команды:

```
while <условие>:
    тело цикла
```

Строка `while <условие>`: является **заголовком цикла**. Эту строку можно прочитать следующим образом: «Пока верно условие, делай...». Команды тела цикла записываются со сдвигом относительно заголовка цикла.

### 14.3. Использование команды *цикл с предусловием* для исполнителя *Робот*

**Пример 14.5.** Написать программу для решения следующей задачи. *Робот* находится в верхнем левом углу поля. Снизу от *Робота* вдоль всего поля расположена горизонтальная линия с проходом в одну нижнюю клетку. Составить алгоритм, выполнив который, *Робот* сможет пройти через проход и закрасить клетку. Расположение прохода заранее неизвестно.

Нам известно, что проход не ограничен стеной снизу. *Робот* может двигаться вправо до тех пор, пока внизу есть стена.

**Пока** снизу стена **повторять**

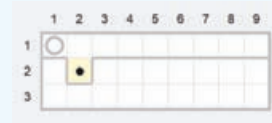
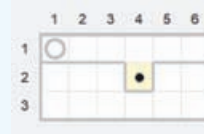
Вправо

*Робот* остановится в той клетке, у которой снизу нет стены. После этого *Робот* должен переместиться вниз и закрасить клетку.

**Пример 14.6.** Написать программу для закраски клеток коридора переменной длины.

В данной задаче нам не известна длина коридора. *Робот* может двигаться, пока справа пусто, и закрашивать клетки:

**Пример 14.5.** Возможные начальные обстановки.

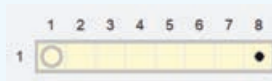


Задачи, в которых могут быть различные обстановки, проверяются на 4–6 различных тестовых примерах. Задача считается решённой, если программа правильно работает для каждого из них.

Программа  
для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_14_5():
    while wall_down():
        move_right()
        move_down()
        fill_cell()
run_tasks()
```

**Пример 14.6.** Одна из возможных начальных обстановок.



**Пример 14.6. Продолжение.**

Программа  
для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_14_4():
    while not wall_right():
        fill_cell()
        move_right()
        fill_cell()
run_tasks()
```

**Пока** нет стены справа **повторять**  
Закрасить  
Вправо

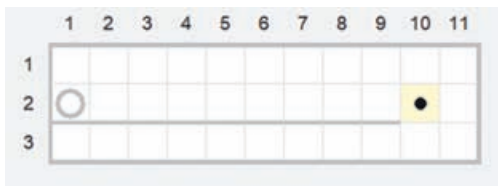
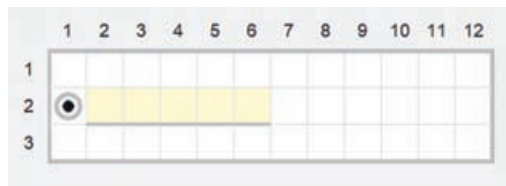
После прохождения всего коридора *Робот* должен закрасить последнюю клетку. Это действие происходит после выполнения цикла (команда `fill_cell()`), так как для последней клетки условие «справа пусто» уже не выполняется — там расположена стена.



1. Что понимают под условием для исполнителя?
2. В каких случаях в алгоритмах используется команда *цикл с предусловием*?
3. Как работает *цикл «пока»*?
4. Как записывается команда *цикл с предусловием* на языке Python?
5. Когда возникает ситуация зацикливания?

**Упражнения**

1. Напишите программы для решения задач `upr_14_1_a` и `upr_14_1_b` из встроенного задачника. Обращайте внимание на начальное и конечное положение *Робота*.

`upr_14_1_a``upr_14_1_b`

2 Для исполнителя *Робот* был написан следующий алгоритм:

```
from pyrob.api import *

@task
def upr_14_2():
    while not wall_right():
        fill_cell()
        move_down()
        move_right()
        fill_cell()
        move_up()
        move_right()

run_tasks()
```

Нарисуйте в тетради результат работы алгоритма. Какими должны быть размеры поля, чтобы *Робот* не врезался в стену? Определите начальное положение *Робота*.

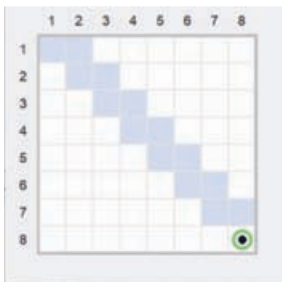
3 Составьте алгоритм, выполнив который, *Робот* нарисует «узор» из упражнения 2 вдоль левого края поля исполнителя (задача `upr_14_3`). Каким должен быть вертикальный размер поля исполнителя?

4 *Робот* находится на квадратном поле неизвестного размера. Начальное положение *Робота* — верхний левый угол. Составьте и выполните алгоритм, по которому *Робот* переместится из начального положения в нижний правый угол и закрасит все клетки своего пути. На каком (каких) из рисунков изображено решение этой задачи? Поясните.

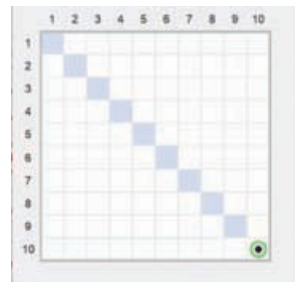
а)



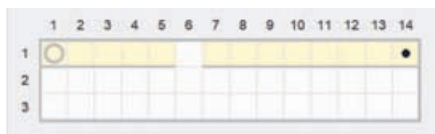
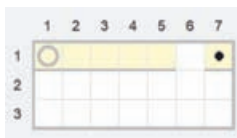
б)



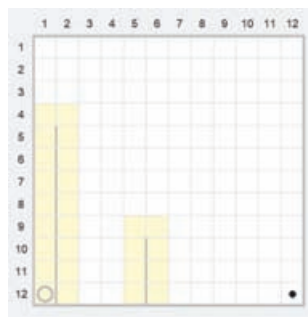
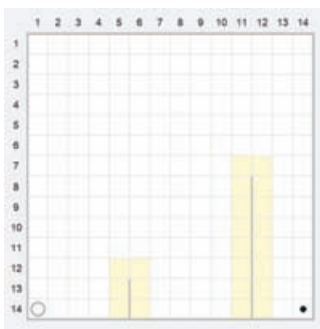
в)



5 На поле *Робота* размещён «забор» — горизонтальная стена. Забор нужно «покрасить» — написать программу для закрашивания всех клеток сверху стены (задача `upr_14_5`). В «заборе» могут быть одни «ворота» — клетка без линий. Длина «забора» и расположение «ворот» неизвестны.



6\* Решите задачу `upr_14_6`. Используйте вспомогательный алгоритм для обхода каждой из двух стен, расположенных на поле *Робота*.



## § 15. Алгоритмическая конструкция ветвление

**Пример 15.1.** Выбор обуви весной в зависимости от погоды.

**Если** на улице дождь, **то**  
надеть резиновые сапоги  
**Иначе**  
надеть туфли

В данном примере в текущий момент времени может быть выполнена только одна команда из двух: или надеть сапоги, или надеть туфли.

### 15.1. Команда ветвление

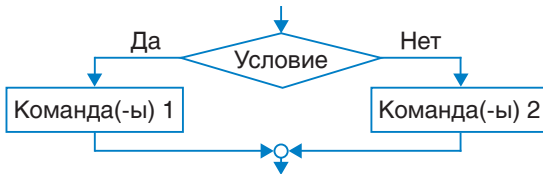
Довольно часто на поставленный вопрос человек получает ответ «да» или «нет». В зависимости от ответа человек определяет свои действия и выполняет одну или другую «команду» или группу «команд» (пример 15.1).

Роботы и другие технические устройства также могут выпол-

нять различные действия в зависимости от условий. Если условие истинно (на вопрос получен ответ «да»), то выполняется один набор действий, если ложно, то другой.

**Алгоритмическая конструкция ветвление** обеспечивает выполнение одной или другой последовательности команд в зависимости от истинности или ложности некоторого условия.

Команда *ветвление* может изображаться на блок-схеме следующим образом:



Принцип работы команды *ветвление* описан в примере 15.2.

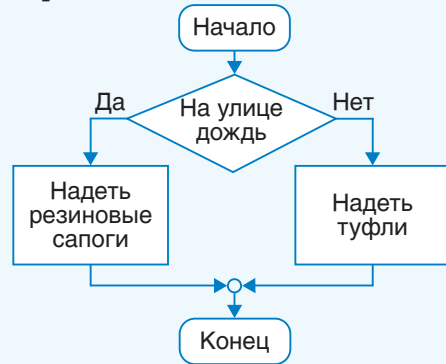
В языке программирования Python для записи конструкции *ветвление* используется команда `if`. Формат записи команды:

```
if <условие>:
    команда(-ы) 1
else:
    команда(-ы) 2
```

Строка `if <условие>`: является заголовком ветвления. Эту строку можно прочитать так:

### Пример 15.1. Продолжение.

Блок-схема данного алгоритма будет выглядеть следующим образом:



Понятие ветвления используется в различных сферах человеческой деятельности.



В ботанике под ветвлением побегов понимают процесс увеличения числа побегов у растений.



Ветвления используются в дорожной разметке и картографии.

При употреблении термина в переносном смысле под ветвлением понимают наличие нескольких путей, направлений, сюжетных линий и т. д.

**Пример 15.2.** Принцип работы команды *ветвление*.

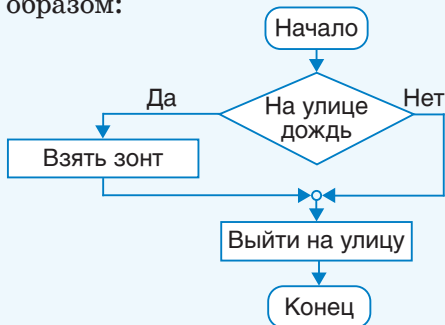
Выполнение конструкции начинается с проверки условия. Если условие истинно, то выполняется последовательность команд 1, если условие ложно, то выполняется последовательность команд 2. При такой организации алгоритма может выполняться **только одна** из двух последовательностей команд. Другая последовательность команд будет проигнорирована.

**Пример 15.3.** Выход на улицу осенью.

**Если** на улице дождь, **то**  
взять зонт  
**выйти** на улицу

В данном примере используется сокращённая форма команды *ветвление*. Если условие истинно, то выполняется команда *взять зонт*. Если условие ложно, то никаких действий не происходит. Команда *выйти на улицу* выполняется всегда, независимо от истинности или ложности условия.

Блок-схема данного алгоритма будет выглядеть следующим образом:

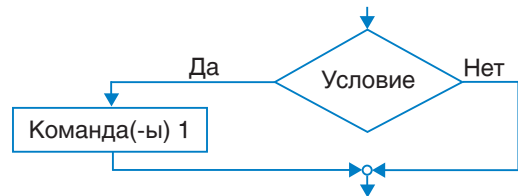


«Если условие верно, то...». В следующей строке со сдвигом записывается последовательность команд 1. После слова *else* в следующей строке со сдвигом записывается последовательность команд 2. Обратите внимание на то, что символ «:» ставится как в конце заголовка ветвления, так и после слова *else*.

Команда *ветвление* может быть записана в **полной** или **сокращённой** формах.

Полная форма команды *ветвления* организует выполнение двух разных наборов команд, из которых выполняется только один. В сокращённой форме один из наборов команд отсутствует. Если отсутствует набор команд, соответствующий ситуации, когда условие ложно, то никакие действия не выполняются (пример 15.3).

Блок-схема сокращённой формы *ветвление*:



На языке программирования Python команда записывается следующим образом:

```
if <условие>:
    команда(-ы) 1
```

В качестве последовательности команда(-ы) 1 и/или 2 может выступать другая команда ветвления. То есть, если истинно (ложно) одно условие, то может потребоваться проверить другое условие. Такие конструкции образуют **вложенные ветвления** (пример 15.4).

В языке Python для записи вложенных ветвлений (в случае ложности первого условия) можно применять конструкцию `elif`. Запись конструкции:

```
if <условие1>:
    команды 1
elif <условие2>:
    команды 2
else:
    команды 3
```

Ключевое слово `elif` в записи команды по существу заменяет два других: `else` и `if`. Блоков `elif` может быть несколько в том случае, если необходимо проверить не два условия, а больше.

## 15.2. Составные условия

В качестве условия в алгоритмах с циклами и ветвлениями используется любое понятное исполнителю этого алгоритма

**Пример 15.4.** Имеется три монеты, среди которых одна фальшивая. Известно, что фальшивая монета легче настоящих монет. Как найти фальшивую монету за минимальное число взвешиваний на чашечных весах без гирь? За какое количество взвешиваний можно определить фальшивую монету?

Представим словесное описание алгоритма решения этой задачи.

Положить на каждую чашу весов по одной монете (монета 1 и монета 2)

**Если** весы в равновесии, **то**  
фальшивая монета 3

**Иначе**

**Если** монета 1 тяжелее, **то**  
фальшивая монета 2

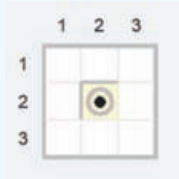
**Иначе**

фальшивая монета 1

За одно взвешивание мы можем определить фальшивую монету.



**Пример 15.5.** Начальная обстановка поля *Робота*.



Первое условие состоит из двух простых:  $A = \text{wall\_left}()$  и  $B = \text{cell\_is\_filled}()$ . Условие может быть записано как  $A \text{ И } B$ . Это условие верно только тогда, когда верны и  $A$ , и  $B$ . Условие  $A = 1$  (истинно), условие  $B = 1$  (истинно), условие  $A \text{ И } B = 1$  (истинно).

Второе условие может быть записано как  $A \text{ ИЛИ } B$ , где  $A = \text{wall\_up}()$ ,  $B = \text{wall\_down}()$ . Условие  $A = 1$ , условие  $B = 0$ . Значит, условие  $A \text{ ИЛИ } B = 1$  (истинно).

В третьем условии операция `not` отрицает составное условие `wall_right()` or `not wall_up()`. Условие может быть записано как **НЕ** ( $A \text{ ИЛИ НЕ } B$ ). Соблюдая порядок выполнения логических операций, определим истинность или ложность данного условия:

- 1) **НЕ**  $B = 0$ , так как  $B = 1$ ;
- 2)  $A \text{ ИЛИ НЕ } B = 0$ , так как  $A = 0$ ;
- 3) **НЕ** ( $A \text{ ИЛИ НЕ } B$ ) = 1.

Для других обстановок *Робота* истинность или ложность приведённых составных условий может быть другой.

высказывание, которое может быть либо истинным, либо ложным.

Все условия, с которыми нам приходилось до сих пор встречаться при составлении алгоритмов для *Робота*, были простыми высказываниями. Для исполнителя *Робот* можно строить и составные условия. Составное условие для *Робота*, так же как и для любого другого исполнителя, образуется из нескольких простых условий, соединённых друг с другом логическими операциями.

С логическими операциями над высказываниями вы уже знакомы. В Python используют следующие логические операции:

| Логическая операция | Запись в Python  |
|---------------------|------------------|
| <b>НЕ</b>           | <code>not</code> |
| <b>И</b>            | <code>and</code> |
| <b>ИЛИ</b>          | <code>or</code>  |

Истинность или ложность составного условия для исполнителя *Робот* определяется для конкретной обстановки.

**Пример 15.5.** Проверить для *Робота* следующие составные условия:

1. `wall_left() and cell_is_filled()`
2. `wall_up() or wall_down()`
3. `not (wall_right() or not wall_up())`

### 15.3. Использование команды ветвление для исполнителя Робот

**Пример 15.6.** Робот должен закрасить клетку, которая находится за стеной. В зависимости от обстановки обход стены может осуществляться по-разному.

В начале Робот должен сдвинуться вправо. Если стена снизу, то сверху свободно и можно обойти стену сверху, в противном случае Робот обходит стену снизу. После обхода стены Робот закрасивает клетку. Алгоритм можно записать следующим образом:

Вправо

**Если** стена снизу, **то**

Вверх

Вправо

Вниз

**Иначе**

Вниз

Вправо

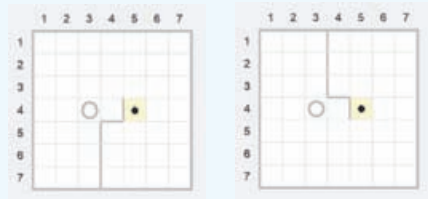
Вверх

Закрасить

Для решения этой задачи использована полная форма команды ветвление.

**Пример 15.7.** Робот находится на неизвестной клетке поля без линий. Он должен закрасить клетку слева от себя.

**Пример 15.6.** Две разные обстановки поля Робота.



Программа для исполнителя Робот:

```
from pyrob.api import *

@task
def prim_15_6():
    move_right()
    if wall_down():
        move_up()
        move_right()
        move_down()
    else:
        move_down()
        move_right()
        move_up()
    fill_cell()

run_tasks()
```

**Пример 15.7.** Программа для исполнителя Робот.

```
from pyrob.api import *

@task
def prim_15_7():
    if not wall_left():
        move_left()
        fill_cell()

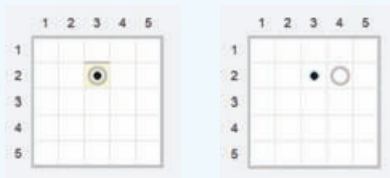
run_tasks()
```

**Пример 15.7. Продолжение.**

Результат работы программы для разных начальных обстановок:

| Начальная обстановка | Результат |
|----------------------|-----------|
|                      |           |
|                      |           |

**Пример 15.8.** Две разные обстановки поля *Робота*.



Программа для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_15_8():
    if wall_up():
        fill_cell()
    elif wall_down():
        fill_cell()
    else:
        move_left()
run_tasks()
```

*Робот* может переместиться влево для закрашивания клетки только в том случае, если слева нет стены, иначе он не выполнит никаких действий и останется на месте. Поэтому, прежде чем сдвинуться влево, *Робот* должен проверить, свободно ли слева. В том случае, когда *Робот* находится возле левой границы поля, он ничего не делает, оставаясь на своём месте.

Результат работы данной программы зависит от начального положения *Робота*. Для решения этой задачи использована сокращённая форма команды *ветвление*.

**Пример 15.8.** *Робот* находится на произвольной, не прилегающей к границе, клетке поля. Вокруг клетки сверху или снизу могут быть стены. Он должен закрасить клетку, на которой находится, если у неё есть хоть одна стена, иначе *Робот* должен сдвинуться влево.

Программа для решения этой задачи может быть записана двумя способами:

1) с использованием конструкции `elif`;

2) с использованием составного условия.

В первом случае *Робот* проверяет условие «стена сверху» и в случае его ложности переходит на проверку условия «стена снизу». Во втором случае используется составное условие: «стена сверху ИЛИ стена снизу».

**Пример 15.8. Продолжение.**

Исполняемый фрагмент программы для второго способа:

```
if wall_up or wall_down():
    fill_cell()
else:
    move_left()
```



1. Что такое алгоритмическая конструкция *ветвление*?
2. Из чего состоит заголовок команды *ветвление*?
3. Чем отличается форма записи полной конструкции *ветвление* от сокращённой?
4. Какая конструкция может использоваться для записи вложенных ветвлений?
5. Что такое составное условие?
6. Какие логические операции можно использовать для записи составных условий?
7. Как определить истинность составного условия?



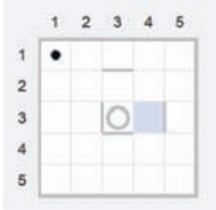
## Упражнения

- 1 Выделите алгоритмическую конструкцию *ветвление* в отрывке из поэмы А. С. Пушкина «Руслан и Людмила» и изобразите эту конструкцию с помощью блок-схемы.

У Лукоморья дуб зелёный;  
 Златая цепь на дубе том:  
 И днём и ночью кот учёный  
 Всё ходит по цепи кругом;  
 Идёт направо — песнь заводит,  
 Налево — сказку говорит.  
 Там чудеса: там леший бродит,  
 Русалка на ветвях сидит...<sup>1</sup>

<sup>1</sup> Пушкин А. С. Руслан и Людмила : поэма. М., 1996. С. 5.

2 Определите, какие из составных условий истинны, а какие ложны для заданной обстановки поля *Робота*.

| Начальная обстановка  | Условия   |
|---|---|
|  | <pre>wall_left() or cell_is_filled() wall_up() and wall_down() not cell_is_filled() and not wall_right() not (wall_up() or not wall_right()) wall_down() and cell_is_filled() (wall_up() or wall_down()) and not wall_right()</pre> |

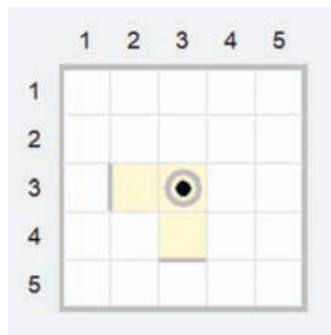
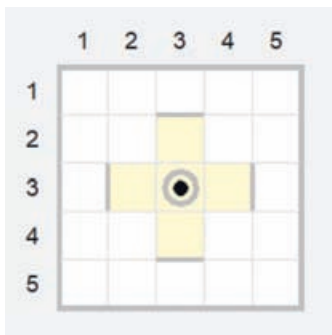
3 В каждом из условий упражнения 2 заменили операцию `and` на `or`, а операцию `or` на `and`. Изменится ли истинность приведённых условий?

4 Для каждого из ложных условий упражнения 2 придумайте обстановку поля *Робота*, в которой данное условие будет истинным, а для каждого истинного — обстановку, в которой условие будет ложным.

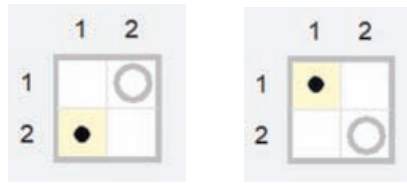
5 Решите задачи `upr_15_5_1` и `upr_15_5_2`. Эти задачи аналогичны примеру 15.7 на с. 111. В задаче `upr_15_5_1` *Робот* должен закрасить клетку справа, а в задаче `upr_15_5_2` — снизу.

6\* Решите задачи `upr_15_6_1` и `upr_15_6_2`.

1. Клетка, на которой находится *Робот*, должна быть закрашена. Остальные клетки закрашиваются только тогда, когда есть стена с соответствующей стороны.



2. *Робот* находится в какой-то клетке поля размером  $2 \times 2$ . Он должен закрасить клетку в углу, противоположном начальному.



## § 16. Использование основных алгоритмических конструкций для исполнителя *Робот*

Последовательное выполнение команд в программе определяется структурой *следование*. Для организации повторяющихся действий в алгоритме используется команда *цикл*. Команда *ветвление* позволяет выполнять одну или другую последовательность команд в зависимости от истинности условия.

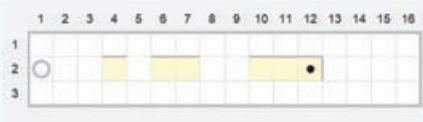
*Следование, цикл и ветвление* — базовые алгоритмические конструкции. Используя эти конструкции как элементы некоего «конструктора», можно составлять и разрабатывать любые алгоритмы.

Команды *цикл* и *ветвление* управляют порядком выполнения других команд в программе и относятся к командам управления. Использование алгоритмической

Перед человеком постоянно возникают разнообразные задачи, для которых существуют различные алгоритмы решения. При всём многообразии алгоритмов для их записи достаточно трёх алгоритмических конструкций (структур): *следование, цикл, ветвление*.

Это положение было выдвинуто в середине 1970-х гг. учёным Эдсгером Вибе Дейкстрой (1930–2002 гг.). Его труды оказали влияние на развитие информатики и информационных технологий. Дейкстра являлся одним из разработчиков концепции структурного программирования, участвовал в разработке языка программирования Алгол. Известен своими разработками в области математической логики и теории графов.

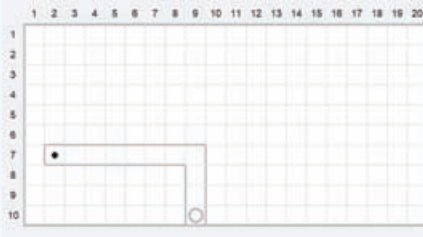
**Пример 16.1.** Одна из возможных начальных обстановок.



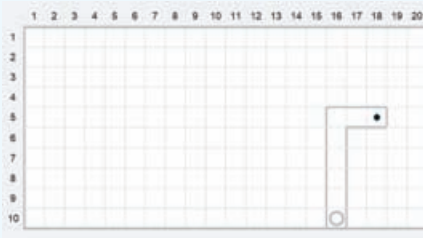
Программа  
для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_16_1():
    while not wall_right():
        if wall_up():
            fill_cell()
            move_right()
        if wall_up():
            fill_cell()
run_tasks()
```

**Пример 16.2.** Одна из возможных начальных обстановок.



Другая обстановка:



конструкции *следование* предполагает отсутствие управляющих конструкций.

Рассмотрим примеры алгоритмов, содержащих несколько алгоритмических конструкций.

**Пример 16.1.** Написать программу для закрашивания некоторых клеток на поле. Окончание движения *Робота* определяет стена справа. По пути нужно закрасить те клетки, над которыми есть стена.

Для решения задачи *Робот* при движении вправо должен проверять каждую клетку на своём пути. Если условие «сверху стена» выполняется, то *Робот* закрашивает эту клетку. После проверки клетки *Робот* сдвигается вправо. Эти действия выполняются в цикле, пока справа нет стены.

После цикла необходима команда *ветвление*, так как для крайней клетки поля условие «нет стены справа» является ложным и клетка в цикле не закрашивается.

В этой задаче внутри конструкции цикла используется конструкция *ветвление*.

**Пример 16.2.** *Робот* должен дойти до конца «коридора» переменного размера. В «коридоре» может быть поворот влево или вправо.

Для решения задачи *Робот* сначала перемещается вверх, до тех пор, пока истинно условие «нет стены сверху». Стена, появившаяся сверху, означает, что начался поворот «коридора». Если слева нет стены, то в «коридоре» надо поворачивать влево, иначе в «коридоре» *Робот* поворачивает вправо. Дальше он двигается в выбранном направлении, пока не дойдёт до стены.

При решении данной задачи используется сначала конструкция *цикл*, а затем конструкция *ветвление*. В команде *ветвление* каждая последовательность команд представляет собой цикл.

**Пример 16.3\*.** *Роботу* необходимо переместиться из верхнего левого угла поля в нижний левый угол. При этом на поле присутствуют стены. *Робот* начинает движение вправо до правой границы поля и спускается на одну клетку вниз. Затем движение продолжается влево до левой границы поля, после — спуск на одну клетку. Эти действия *Робот* должен повторить 4 раза.

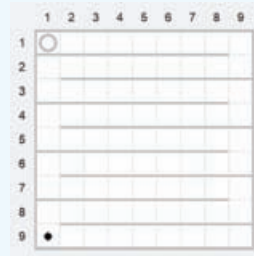
В данной задаче внутри цикла с параметром используются два других цикла с предусловием.

### Пример 16.2. Продолжение.

Программа  
для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_16_2():
    while not wall_up():
        move_up()
    if wall_left():
        while not wall_right():
            move_right()
    else:
        while not wall_left():
            move_left()
run_tasks()
```

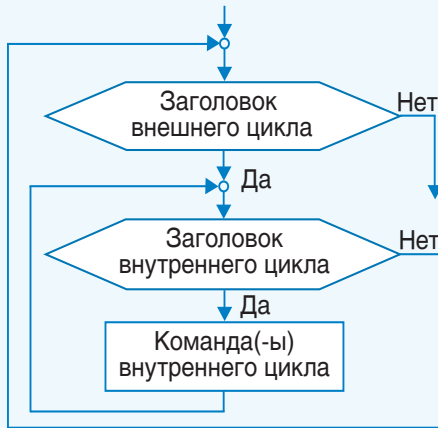
### Пример 16.3\*. Начальная обстановка.



Программа  
для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_16_3():
    for i in range(4):
        while wall_down():
            move_right()
            move_down()
        while wall_down():
            move_left()
            move_down()
run_tasks()
```

**Пример 16.4.** Блок-схема вложенных циклов.



Структуру, когда внутри одного цикла выполняется другой цикл, называют **вложенным циклом**.

Один из возможных вариантов вложенных циклов представлен на блок-схеме в примере 16.4.

Как видно из примеров, базовые алгоритмические структуры можно комбинировать друг с другом так, как этого требует алгоритм решения задачи.



1. Назовите базовые алгоритмические конструкции.
2. Какие базовые алгоритмические конструкции относят к элементам управления?
3. Приведите примеры использования базовых алгоритмических конструкций.
- 4\*. Что такое вложенный цикл?



## Упражнения

**1** Объясните, какие алгоритмические конструкции используются в приведённых ниже программах. Нарисуйте блок-схемы данных алгоритмов. Предложите пример начальной обстановки, в которой алгоритм выполнится корректно.

а)

```

from pyrob.api import *
@task
def upr_16_1_a():
    while wall_left():
        fill_cell()
        move_down()
run_tasks()
  
```

б)

```

from pyrob.api import *
@task
def upr_16_1_b():
    while cell_is_filled():
        if not wall_left():
            move_left()
run_tasks()
  
```

2 Для решения задачи `upr_16_2` Миша написал программу, но она работает неправильно. Какие ошибки допустил Миша?



```
from pyrob.api import *
@task
def upr_16_2():
    while wall_right():
        if wall_up() or wall_down():
            fill_cell()
        move_right()
    if wall_up() and wall_down():
        fill_cell()
run_tasks()
```

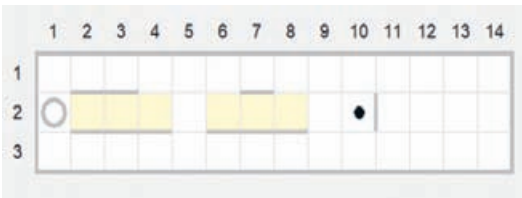
3 Заполните пропуски (многоточие на жёлтом фоне) в программе решения задачи `upr_16_4` так, чтобы она работала верно.



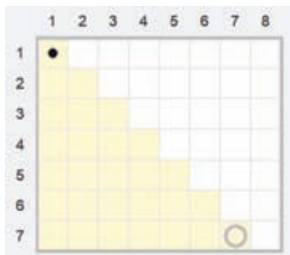
```
from pyrob.api import *
@task
def upr_16_3():
    for i in range (4):
        fill_cell()
        while ...:
            move_right()
            fill_cell()
        move_down()
        fill_cell()
        while ...:
            move_left()
            fill_cell()
        ...
    while not wall_right():
        move_right()
        fill_cell()
run_tasks()
```

В строке поля *Робота* может быть произвольное количество клеток.

4 Решите задачу `upr_16_4`, используя внутри цикла команду *ветвление*.



5\* Решите задачу `upr_16_5`, используя вложенные циклы. Размер поля заранее не известен.



6\* Придумайте задачу для исполнителя *Робот*, в которой будут использоваться различные алгоритмические конструкции.

## § 17. Язык программирования Python

**Компьютер** (англ. computer — *вычислитель*) — устройство (или система), способное выполнять заданную чётко определённую изменяемую последовательность операций. Чаще всего это операции численных расчётов.

Гвидо ван Россум приступил к созданию языка Python в декабре 1989 г. в центре математики и информатики в Нидерландах. Ван Россум является основным автором Python. Он продолжал выполнять центральную роль в принятии решений относительно развития языка вплоть до 12 июля 2018 г.

Желание упростить расчёты присуще человеку с древних времён. Создавая различные приспособления для счёта, человек прошёл долгий путь. Современный компьютер способен выполнять сотни миллионов операций в секунду. Для решения вычислительных задач требуется сначала составить алгоритм, а затем записать его в виде программы, используя какой-либо язык программирования.

**Язык программирования** устанавливает набор правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под её управлением.

Сегодня язык Python является одним из самых популярных и востребованных языков программирования. Существует большое количество сред программирования с поддержкой языка Python: IDLE, PyCharm, Spyder, Thonny, Eclipse + PyDev, Visual Studio и др. В учебном курсе используется среда IDLE, с которой вы работали, знакомясь с учебными компьютерными исполнителями.

### 17.1. Команда вывода

Демонстрировать работу любой программы имеет смысл только тогда, когда она выводит какую-либо информацию.

**Пример 17.1.** Написать программу для вывода слов «Привет, мир!».

```
print('Привет, мир!')
```

Результат работы программы отражается в главном окне среды IDLE.

Команда `print()` предназначена для **вывода данных**. Она является функцией, аргументы которой — выводимые значения.

Аргументами функции `print` могут быть разделители (пример 17.2).

**Пример 17.1.** Окно среды IDLE с результатом работы программы.

Результат работы программы

По традиции, начавшейся в 1978 г. с примера из книги Б. Кернигана и Д. Ритчи «Язык программирования Си», первая программа на любом языке программирования должна просто выводить на экран приветствие миру.

**Пример 17.2.** Аргументы команды `print()` для управления разделителями.

`sep` — задаёт текст, который будет служить разделителем при выводе различных значений в одной команде `print()`. По умолчанию — пробел.

`end` — задаёт текст, который появится на экране после вывода всех значений одной команды `print()`. По умолчанию — перевод строки.

В программе перевод строки задаётся как `'\n'`. Если использовать `sep = '\n'`, то каждое выводимое значение одной команды `print()` будет расположено в новой строке.

Аргументы, управляющие выводом, могут использоваться в команде `print()` по одному или оба сразу.

У функции `print` есть ещё два необязательных аргумента: `file` (определяет, куда выводятся данные, по умолчанию — на экран) и `flush` (определяет, нужно ли принудительно очищать поток вывода, по умолчанию — `false`).

### Пример 17.3. Текст программы.

```
print('Привет!',
      'Я компьютер!!!')
print('Я умею выполнять',
      end = ' ')
print('программы!', 'Здесь',
      sep = '\n', end = ' ')
print('ты', 'написал свою',
      end = ' ')
print('первую программу,',
      end = ' ')
print('и я её выполнил.')
print('Сейчас на экране',
      end = ' - ')
print('её результат.')
print()
print('Ура!', 'Ура!', 'Ура!',
      sep = '!')
```

Результат работы программы:

```
===== RESTART: C:/prog_7k1/17_3.py =====
Привет! Я компьютер!!!
Я умею выполнять программы!
Здесь ты написал свою первую программу, и я её выполнил.
Сейчас на экране - её результат.
Ура!Ура!Ура!
```

Текст, который нужно вывести на экран, заключают в кавычки. Кавычки могут быть одинарными или двойными. Текст выводится на экран в том виде, в котором он записан в программе. Его можно записать как на русском, так и на любом другом языке. Текстом может быть произвольный набор символов.

В программе может быть несколько команд вывода. После выполнения команды вывода курсор переводится на следующую строку. В одной команде можно указать вывод нескольких значений, которые отделяются друг от друга запятыми. Если функция `print()` используется без аргументов, то будет выведена пустая строка.

**Пример 17.3.** Вывести на экран компьютера следующий текст, используя различные значения разделителей: «Привет! Я компьютер!!! Я умею выполнять программы! Здесь ты написал свою первую программу, и я её выполнил. Сейчас на экране — её результат. Ура! Ура! Ура!».

Используя сочетания аргументов `sep` и `end`, можно управлять размещением текста на экране.

Текст в команде `print()`, записанный в кавычках, не анализируется. Если кавычки опустить, то производится анализ тех данных, которые записаны в скобках. Например, если в скобках написать арифметическое выражение, то сначала вычисляется его значение, а затем выводится результат.

**Пример 17.4.** Посчитать значение выражения  $2 + 2 \cdot 2$ .

Если в команде `print()` записать выражение в кавычках, то на экран будет выведено само выражение, при отсутствии кавычек — значение данного выражения.

## 17.2. Понятие типа данных

На практике редко приходится писать программы, которые решают только одну задачу. Обычно программы создаются для решения целого класса задач, которые можно сформулировать в общем виде.

С такими задачами вы уже сталкивались в курсе математики. Например, решение задачи «Найти площадь прямоугольника» можно записать так:  $S = a \cdot b$ , где переменные  $a$  и  $b$  обозначают соответственно длину и ширину прямоугольника, а  $S$  — площадь.

**Пример 17.4.** Текст программы.

```
print('2 + 2 * 2 =')
print(2 + 2 * 2)
```

Результат работы программы:

```
2 + 2 * 2 =
6
```

Для вывода результата в одной строке программу нужно записать так:

```
print('2 + 2 * 2 =', end = ' ')
print(2 + 2 * 2)
```

Или так:

```
print('2 + 2 * 2 =', 2 + 2 * 2)
```

Результат работы программы:

```
2 + 2 * 2 = 6
```

Программисты ЭВМ до начала 1950-х гг. при создании программ пользовались машинным кодом. Запись программы на машинном коде состояла из единиц и нулей. Типы данных не использовались. Машинный код принято считать языком программирования первого поколения.

Первым языком программирования, в котором появилась возможность создавать переменные, считается Ассемблер. В этом языке вместо машинного кода стали использовать команды, записанные текстом. Ассемблер относится к языкам программирования второго поколения.

В 1957 г. появился язык Фортан, открывший эру языков программирования третьего поколения. Он позволил использовать разные числовые типы данных, необходимые для сложных расчётов: целые, вещественные (действительные) и комплексные (комплексные числа изучаются на факультативах по математике в старших классах).

Дальнейшее развитие языков программирования позволило добавить возможность работы с другими типами данных. Нынешние языки программирования поддерживают возможность работы с большим количеством разных типов данных.

**Пример 17.5.** Числовые типы данных.

Язык Python поддерживает работу с целыми и рациональными числами. Значения переменных могут определяться так же, как в математике.

```
a = 2
b_1 = 7.5
radius = 12.0
```

Переменная `a` будет определена как целая, а переменные `b_1` и `radius` — как рациональные. Целая часть дробного числа отделяется от дробной части точкой.

Зная эту формулу, можно найти площадь любого прямоугольника.

В программировании для решения задач в общем виде также используют переменные. Для работы компьютера с переменными они должны храниться в его памяти.

Информацию, представленную в формализованном виде и пригодную для обработки на компьютере, называют **данными**.

**Переменная** в программировании — это именованная ячейка памяти, хранящая значение переменной.

Компьютер может обрабатывать данные разных типов: целые и рациональные числа, строки и др.

**Тип данных** определяет способ их хранения в памяти компьютера, диапазон возможных значений и операции, которые можно выполнять с этим типом данных.

Каждая переменная задаётся своим именем. Для обозначения имени переменной используют буквы латинского алфавита, арабские цифры и знак «`_`».

Заглавные и строчные буквы считаются различными.

При использовании переменной выделяется память для хранения значения этой переменной. В процессе выполнения программы значение переменной может изменяться. Тип переменной в языке Python определяется по её значению (пример 17.5 на с. 124). Это происходит в процессе выполнения программы.

### 17.3. Оператор присваивания

Одной из основных команд для обработки данных в программе является оператор присваивания.

**Оператор присваивания** предназначен для того, чтобы:

- задавать значения переменных;
- вычислять значение арифметического выражения, результат вычисления которого будет записан как значение переменной.

Формат записи оператора присваивания:

<имя переменной> = <выражение>

В записи арифметического выражения используются знаки математических действий: сложения, вычитания, умножения, деления

#### Пример 17.5. Продолжение.

Задавать значения можно и таким образом:

```
a, b_1, radius = 2, 7.5, 12.0
```

Чтобы убедиться, что переменные получили указанные значения, выведем их с помощью команды print():

```
print(a, b_1, radius)
```

Результат работы программы:

```
2 7.5 12.0
```

**Пример 17.6.** Запись оператора присваивания.

Запись оператора присваивания в программе и вывод значений переменных:

```
x = 9
x1 = 3.5
a_1 = 20 * (x + x1) - 32
y = 3
chastnoe = x / y
y = 7 + 2 * y ** 3
print('x =', x)
print('x1 =', x1)
print('a_1 =', a_1)
print('chastnoe =',
      chastnoe)
print('y =', y)
```

Результат работы программы:

```
x = 9
x1 = 3.5
a_1 = 218.0
chastnoe = 3.0
y = 61
```

Если в выражении встречаются переменные разных числовых типов (как целых, так и рациональных), то результат будет рациональным числом (значение переменной  $a_1$ ).

**Пример 17.7.** Запись оператора присваивания на Python для математических выражений:

| Выражение             | Запись на Python                                      |
|-----------------------|---|
| $S = 2(a + b)$        | <code>S = 2 * (a + b)</code>                          |
| $S = a^2$             | <code>S = a * a</code> или<br><code>S = a ** 2</code> |
| $a = \frac{x + y}{3}$ | <code>a = (x + y) / 3</code>                          |

**Пример 17.8\*.** Изменение значения переменной.

В Python допустимы команды присваивания следующего вида:

```
a = a * 2 или a *= 2
```

Смысл команды следующий: из ячейки памяти извлекается значение переменной  $a$ , затем оно умножается на 2, результат записывается в ту же ячейку памяти. Старое значение переменной  $a$  будет потеряно.

Запись оператора присваивания для изменения значения переменной  $b$  следующая:

```
b = b - 3 или b -= 3
```

и возведение в степень. В языке Python им соответствуют следующие символы:

| Математические операции | Запись в Python |
|-------------------------|-----------------|
| + (сложение)            | +               |
| - (вычитание)           | -               |
| · (умножение)           | *               |
| : (деление)             | /               |
| возведение в степень    | **              |

Приоритет выполнения операций соответствует принятому в математике: сначала выполняется возведение в степень, затем умножение и деление, далее — сложение и вычитание. В выражениях для изменения порядка действий можно использовать скобки (примеры 17.6 и 17.7).

Для записи обыкновенной дроби используется знак деления. Знак умножения опускать нельзя. Вычисления, производимые над целыми числами, всегда точные. При вычислениях с рациональными числами результат может быть приближительным.

**Пример 17.8\*.** Записать оператор присваивания, после выполнения которого значение переменной  $a$  увеличится в два раза, а переменной  $b$  уменьшится на 3.

## 17.4. Ввод данных

Начальные значения переменных можно задавать не только с помощью оператора присваивания, но и вводить с клавиатуры. В этом случае текст программы не нужно изменять для нового набора данных.

Команда `input()` предназначена для **ввода данных**. Для того чтобы значение переменной вводилось с клавиатуры, нужно присвоить ей значение функции `input()`.

При использовании в программе команды ввода данных в виде `a = input()` все символы, вводимые с клавиатуры, при выполнении команды воспринимаются как текст (в том числе и числа). Поэтому в команде ввода нужно указать, в какой числовой тип данных мы хотим преобразовать вводимый текст (пример 17.9).

В качестве параметра функции `input()` можно задать текст, который будет служить подсказкой при вводе данных (пример 17.10).

Ввод данных осуществляется в главном окне среды IDLE (пример 17.10). После завершения работы программы в этом же окне будет выведен результат.

### Пример 17.9. Ввод чисел.

Для преобразования в целое число используется функция `int()`, а в рациональное число — функция `float()`.

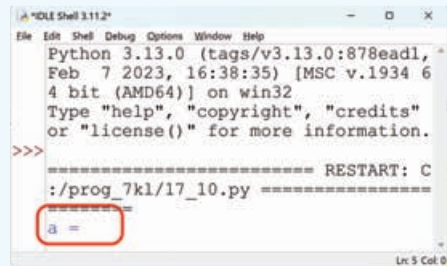
```
x = int(input('целое'))
y = float(input('рациональное'))
```

**Пример 17.10.** Ввести два рациональных числа, найти их сумму.

Текст программы:

```
a = float(input('a = '))
b = float(input('b = '))
s = a + b
print('a + b =', s)
```

После запуска программы в главном окне среды IDLE появится строка `a =`.



В этой строке нужно набрать числовое значение. Обратите внимание на то, что программа выводит текст синим цветом, а значения, вводимые пользователем, отображаются чёрным.

Результат работы программы:

```
===== RESTART: C
:/prog_7kl/17_10.py =====
a = 3.2
b = 4.5
a + b = 7.7
>>>
```



1. Какая команда языка программирования Python предназначена для вывода данных?
2. Как можно изменить разделитель при выводе данных?
3. Что такое переменная?
4. Что определяет тип данных?
5. Для чего используется команда присваивания?
6. Какая команда языка программирования Python предназначена для ввода данных?



### Упражнения

1 Выполните следующие задания для программы из примера 17.3 на с. 122 (файл с программой можно скачать).

1. Замените все аргументы `sep` на `end` и выполните программу. Что произошло? Объясните почему.

2. Как изменится результат работы программы, если в исходном тексте заменить все аргументы `end` на `sep`?

3. Измените программу так, чтобы текст на экране выглядел следующим образом:

Привет! Я компьютер!!! Я умею выполнять программы!

Ты сегодня написал свою первую программу!!!

Я выполнил твою программу. Посмотри на результат!

2 Для примера 17.4 на с. 123 внесите изменения в программу так, чтобы действия выполнялись в том порядке, в котором записаны, то есть сначала сложение, а потом умножение.

3 Составьте программу для определения возраста пользователя через 5 лет и выведите результат. Возраст пользователя вводится в годах.

4\* Составьте программу, которая позволяет ввести два числа  $a$  и  $b$ , затем первое число уменьшает в два раза, а второе увеличивает на 30. Выведите изменённые значения переменных.

5 Напишите программу для вычисления значения числового выражения.

$$1) 23 + 45 \cdot 11 - 15. \quad 2) \frac{37 + 2^5}{41}. \quad 3) \frac{5638 - 2347}{49} + \frac{12^3 \cdot (7 + 56)}{455}.$$

## § 18. Организация вычислений

При решении любой задачи человеку приходится выполнять следующие действия:

- определение исходных данных (что дано в задаче);
- определение результатов (что нужно получить);
- обработка исходных данных в соответствии с известными правилами так, чтобы получить результат;
- проверка и анализ полученного результата.

Решения задач по физике и химии принято оформлять соответствующим образом (пример 18.1).

Применив данные правила к решению задачи по программированию, получим следующие этапы:

- I. Определение исходных данных.
- II. Определение результатов.
- III. Составление алгоритма решения задачи.
- IV. Определение типов данных для переменных, используемых при реализации алгоритма.
- V. Написание программы.
- VI. Тестирование программы.
- VII. Анализ результатов.

**Пример 18.1.** Оформление задач.

Физика:

|  |  |
|--|--|
| <p>Дано:</p> $\frac{\rho_x}{\rho_y} = k$<br>$\frac{V_x}{V_y} = n$<br>$\frac{F_x}{F_y} = ?$ | <p>Решение</p> <p>Силы тяжести, действующие на бруски, равны:<br/> <math>F_x = gm_x</math>; <math>F_y = gm_y</math>.</p> <p>Массы брусков равны: <math>m_x = \rho_x V_x</math>; <math>m_y = \rho_y V_y</math>.</p> <p>Отношение сил: <math>\frac{F_x}{F_y} = \frac{\rho_x V_x}{\rho_y V_y} = \frac{k}{n}</math>.</p> <p>Ответ: <math>\frac{F_x}{F_y} = \frac{k}{n}</math>.</p> |
|--|--|

Химия:

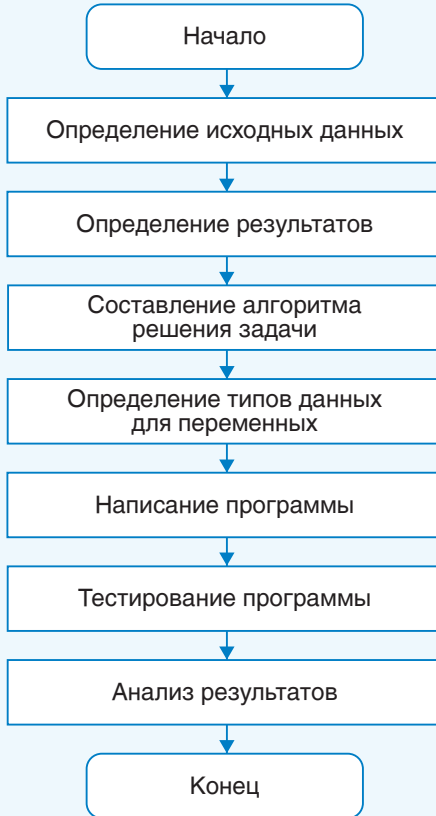
|  |  |
|--|--|
| <p>Дано:</p> $n(\text{H}_2\text{SO}_4) = 3 \text{ моль}$<br>$N(\text{H}_2\text{SO}_4) = ?$ | <p>Решение</p> $N = N_A \cdot n$<br>$N(\text{H}_2\text{SO}_4) = N_A \cdot n(\text{H}_2\text{SO}_4) =$<br>$= 6,02 \cdot 10^{23} \text{ молекул/моль} \cdot 3 \text{ моль} =$<br>$= 18,06 \cdot 10^{23} \text{ молекул}$ . <p>Ответ: в серной кислоте химическим количеством 3 моль число молекул составляет <math>18,06 \cdot 10^{23}</math>.</p> |
|--|--|

Слева записывается, что дано (определение исходных данных) и что нужно получить (определение результатов), справа — последовательность действий, приводящая к решению задачи (обработка исходных данных). Проверка и анализ полученного результата проводятся вручную.

В программировании, в отличие от физики, химии и др., проверка правильности работы программы производится с помощью компьютерного тестирования.

Для анализа данных можно, например, произвести вычисления на калькуляторе. В промышленном программировании для проверки правильности работы одной программы часто пишется другая программа.

**Пример 18.2.** Этапы решения задачи по программированию.



**Пример 18.3.** Решение.  
V. Программа:

```

x = float(input('x = '))
y = float(input('y = '))
z = float(input('z = '))
a = (2 * x + y - z) / (3 + x * x)
print('a =', a)
  
```

В примере 18.2 описанные этапы отображены в виде блок-схемы.

**Тестирование программы** — это проверка правильности работы программы при разных наборах исходных данных.

Анализ данных предполагает проверку правильности результата работы программы. Он может выполняться различными способами: вручную, с помощью технических средств, программно.

Для каждой задачи, рассматриваемой далее, будем описывать все этапы решения.

### 18.1. Вычисление значения арифметического выражения

**Пример 18.3.** Даны  $x$ ,  $y$ ,  $z$ . Написать программу для вычисления значения выражения

$$a = \frac{2x + y - z}{3 + x^2}.$$

Этапы выполнения задания:

I. Определение исходных данных: переменные  $x$ ,  $y$ ,  $z$ .

II. Определение результатов: переменная  $a$ .

III. Алгоритм решения задачи.

1. Ввод исходных данных.
2. Вычисление значения выражения.

3. Вывод результата.

IV. Описание переменных: все переменные, определённые для решения задачи, имеют тип `float`.

В приведённом примере для каждой команды ввода записан текст с пояснениями о том, значение какой переменной нужно вводить.

При написании программ для вычисления значения арифметического выражения часто допускают следующие ошибки (пример 18.4)

## 18.2. Использование языка программирования для решения задач

**Пример 18.5.** Напишите программу для решения геометрической задачи. Дан квадрат с длиной стороны  $a$ . Найдите его площадь и периметр.

Этапы выполнения задания:

I. Определение исходных данных: переменная  $a$  (длина стороны).

II. Определение результатов: переменные  $S$  (площадь) и  $P$  (периметр).

III. Алгоритм решения задачи.

1. Ввод исходных данных.

Вычисление значений площади производится по формуле  $S = a^2$ , а периметра — по формуле  $P = 4a$ .

**Пример 18.3. Продолжение.**

VI. Тестирование программы.

Запустите программу и введите значения:  $x = 2$ ,  $y = 3.1$ ,  $z = 4$ . Проверьте, результат должен быть следующим:

```
x = 2
y = 3.1
z = 4
a = 0.4428571428571428
```

VII. Проверить правильность вычислений можно на калькуляторе. Все данные в программе были определены как рациональные числа. При тестировании допускается ввод и целых значений (переменные  $x$  и  $z$ ), так как целые числа являются подмножеством рациональных.

**Пример 18.4.** Часто допускаемые ошибки.

|                                 |                  |
|---------------------------------|------------------|
| $\frac{2x + 3}{(x - 4)(x + 2)}$ | Пропущен знак *  |
| $(2 + y) / (x * \frac{x}{x})$   | Пропущена скобка |

**Пример 18.5. Решение.**

$$a \quad \begin{cases} S = a^2 \\ P = 4a \end{cases}$$

**Пример 18.5.** *Продолжение.*

V. Программа:

```
a = float(input('a = '))
S = a ** 2
P = 4 * a
print('площадь =', S)
print('периметр =', P)
```

VI. Тестирование программы.

Запустите программу и введите значение  $a = 5.17$ .

Проверьте, результат должен быть следующим:

```
a = 5.17
площадь = 26.7289
периметр = 20.68
```

VII. Правильность вычислений можно проверить на калькуляторе.

**Пример 18.6.** Решение.

V. Программа:

```
s = float(input('s = '))
t = float(input('t = '))
v = s / t
print('скорость =', v)
```

VI. Тестирование программы.

Запустите программу и введите значения  $s = 3550$  и  $t = 4$ .

Проверьте, результат должен быть следующим:

```
s = 3350
t = 4
скорость = 837.5
```

VII. Правильность вычислений можно проверить на калькуляторе.

В программе этим формулам будут соответствовать команды присваивания  $S = a ** 2$  и  $P = 4 * a$ .

2. Вывод результата.

IV. Описание переменных: все переменные имеют тип float.

Обратите внимание на запись операторов присваивания, соответствующих математическим формулам.

**Пример 18.6.** Напишите программу для решения физической задачи. Расстояние между двумя городами  $s$  км. Самолёт пролетает это расстояние за  $t$  ч. Определите скорость самолёта.

Этапы выполнения задания.

I. Определение исходных данных: переменные  $s$  (расстояние) и  $t$  (время).

II. Определение результатов: переменная  $v$  (скорость).

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Согласно формуле расстояния  $s = vt$ . Отсюда выразим  $v$  как  $v = \frac{s}{t}$ .

3. Вывод результата.

IV. Описание переменных: все переменные имеют тип float.

В программе можно использовать **комментарии** — текст, который не анализируется при запуске программы на выполнение.

Текст после символов `##` считается комментарием и выделяется на экране красным цветом. В комментариях удобно записывать условие задачи и пояснения к командам (пример 18.7).

Для того чтобы сделать комментарием сразу несколько строк, нужно их выделить и выполнить команду `Format` → `Comment Out Region` или воспользоваться комбинацией клавиш `Alt + 3`. Команда вставит в начале каждой выделенной строки знаки `##`. Команда `Format` → `Uncomment Region` (`Alt + 4`) превращает комментарии в обычный текст (пример 18.8).

**Пример 18.7.** Программа с комментариями.

```
File Edit Format Run Options Window Help
1 ##Расстояние между двумя
2 ##городами s км.
3 ##Самолет пролетает это расстояние
4 ##за t часов.
5 ##Определите скорость самолета.
6
7 s = float(input('s = '))
8 t = float(input('t = '))
9 #вычисление скорости
10 v = s / t
11 print('скорость =', v)
12
```

**Пример 18.8.** Меню `Format`.

|   | File | Edit | Format                    | Run | Options | Window |
|---|------|------|---------------------------|-----|---------|--------|
| 1 | ##P  |      | Format Paragraph          |     |         | Alt+Q  |
| 2 | ##I  |      | Indent Region             |     |         | Ctrl+] |
| 3 | ##C  |      | Dedent Region             |     |         | Ctrl+[ |
| 4 | ##3  |      | Comment Out Region        |     |         | Alt+3  |
| 5 | ##4  |      | Uncomment Region          |     |         | Alt+4  |
| 6 | ##O  |      | Tabify Region             |     |         | Alt+5  |
| 7 | S =  |      | Untabify Region           |     |         | Alt+6  |
| 8 | t =  |      | Toggle Tabs               |     |         | Alt+T  |
|   |      |      | New Indent Width          |     |         | Alt+U  |
|   |      |      | Strip Trailing Whitespace |     |         |        |



1. Перечислите основные этапы решения задачи по программированию.
2. Что понимают под тестированием программы?
3. Для чего можно использовать комментарии?



### Упражнения

1 Напишите программу для вычисления значения арифметического выражения.

$$1) a = \frac{x + y - z}{x^2 + 2}. \quad 2) a = 5 + \frac{2x - z}{3 + y^3}. \quad 3) a = (1 + z) \frac{x + \frac{y}{x^2 + 4}}{2 + \frac{1}{x^4 + 4}}.$$

2 Напишите программу для решения геометрической задачи.

1. Найдите длину окружности и площадь круга заданного радиуса.

2. Найдите угол при вершине равнобедренного треугольника, если известны величины углов при основании треугольника.

3\*. Найдите углы при основании равнобедренного треугольника, если известен угол при вершине.

3 Напишите программу для решения физической задачи.

1. Велосипедист едет с постоянной скоростью  $v$  км/ч. За сколько минут он проедет путь в  $s$  километров?

2. Давление воды в трубе водопровода на первом этаже здания составляет  $p$  кПа. Определите, на какой высоте находится поверхность воды в водонапорном баке относительно крана, расположенного на первом этаже. Плотность воды  $\rho = 1000$  кг/м<sup>3</sup>, ускорение свободного падения  $g = 10$  м/с<sup>2</sup>.

3\*. Автомобиль проходит первую часть пути длиной  $s_1$  км за  $t_1$  минут, участок пути длиной  $s_2$  км за  $t_2$  минут и, наконец, участок длиной  $s_3$  км за  $t_3$  минут. Найдите среднюю скорость движения автомобиля, выраженную в км/ч.

4 Напишите программу для решения химической задачи.

1. В организме человека на долю атомов кислорода приходится 65 % от массы тела. Найдите массу атомов кислорода для своей массы тела.

2. Рассчитайте массу сахара, который нужно растворить в воде массой 250 г, для получения раствора с массовой долей сахара, равной 5 %.

3\*. Масса одного атома кислорода  $2,656 \cdot 10^{-26}$  кг (это число в программе на языке Python записывается как 2.656E-26, буква E — английская). Определите, сколько атомов кислорода содержится в вашем теле.

## § 19. Реализация алгоритмов работы с целочисленными данными

### 19.1. Целочисленный тип данных

Часто при решении задач используются целые числа. В языке Python целые числа могут быть сколь угодно большими. Для этого типа данных определены следующие операции:

| Математические операции | Запись в Python |
|-------------------------|-----------------|
| + (сложение)            | +               |
| - (вычитание)           | -               |
| · (умножение)           | *               |
| целочисленное деление   | //              |
| нахождение остатка      | %               |
| возведение в степень    | **              |

Для целочисленных данных не определена операция деления (как для рациональных чисел). Если использовать эту операцию, то результат всегда будет рациональным, даже если одно число делится на другое без остатка.

Для организации вычислений с целыми числами определены операции // и %. Значения результатов операций // и % могут отличаться от принятых в математике (пример 19.1).

Чтобы облегчить визуальную оценку величины числа, в Python, начиная с версии 3.6, между цифрами разрешается вставлять одиночные символы подчёркивания. Запись  $a = 1000000$  эквивалентна записи  $a = 1\_000\_000$ . Такую запись можно использовать как при написании текста программы, так и при вводе чисел с клавиатуры.

**Пример 19.1.** Результат операций // и % для чисел разных знаков.

| a   | b  | a // b | a % b |
|-----|----|--------|-------|
| 17  | 3  | 5      | 2     |
| -17 | 3  | -6     | 1     |
| 17  | -3 | -6     | -1    |
| -17 | -3 | 5      | -2    |

При выполнении операции // знак результата определяется так же, как в математике: он положительный, если исходные числа одного знака, и отрицательный, если разного. Значение результата округляется до ближайшего целого в меньшую сторону.

**Пример 19.1.** *Продолжение.*

Результат операции % может быть отрицательным, хотя в математике под остатком понимают неотрицательное число. Если остаток не равен нулю, то знак числа, которое является результатом операции %, определяется знаком делителя. При анализе данных из таблицы можно понять, что

$$a \% b = a - (a // b) * b.$$

**Пример 19.2.** Решение.

V. Программа:

```
a = int(input('a = '))
b = int(input('b = '))
c = a // b
d = a % b
print('целая часть =', c)
print('остаток =', d)
```

VI. Тестирование программы.

Запустите программу и введите значения  $a = 11$  и  $b = 4$ .

Проверьте, результат должен быть следующим:

```
a = 11
b = 4
целая часть = 2
остаток = 3
```

**Пример 19.3.** Решение.

V. Программа:

```
c = int(input('c = '))
#минуты
m = c // 60
#секунды
s = c % 60
print(m, s, sep = ':')
```

Эти операции имеют такой же приоритет, как и операции деления и умножения.

**Пример 19.2.** Даны два целых числа  $a$  и  $b$ . Напишите программу, которая находит целую часть от деления  $a$  на  $b$  и остаток от деления  $a$  на  $b$ .

Этапы выполнения задания.

I. Определение исходных данных: переменные  $a$  и  $b$ .

II. Определение результатов: переменные  $c$  (целочисленное частное) и  $d$  (остаток).

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Целочисленное частное находим как результат операции  $a // b$ , остаток —  $a \% b$ .

3. Вывод результата.

IV. Описание переменных: все переменные имеют тип `int`.

## 19.2. Использование целочисленных данных для решения задач

**Пример 19.3.** Пусть таймер показывает время только в секундах. Написать программу, которая переведёт время в минуты и секунды.

Этапы выполнения задания.

I. Определение исходных данных: переменная  $c$  (время в секундах).

II. Определение результатов: переменные  $m$  (полное количество минут) и  $s$  (остаток секунд).

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Для нахождения полного числа минут нужно найти целую часть от деления исходного числа секунд на 60.

3. Оставшиеся секунды можно найти как остаток от деления исходного числа секунд на 60.

4. Вывод результата.

IV. Описание переменных: все переменные имеют тип `int`.

**Пример 19.4.** Задано положительное двузначное число. Написать программу, которая поменяет местами первую и вторую цифры числа.

Этапы выполнения задания.

I. Определение исходных данных: переменная  $a$  (исходное число).

II. Определение результатов: переменная  $b$  (преобразованное число).

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Для преобразования числа необходимо выполнить следующие действия:

**Пример 19.3. Продолжение.**

VI. Тестирование программы.

Запустите программу и введите значение  $c = 137$ . Результат должен быть следующим:

```
c = 137
2:17
```

Для  $c = 24$  получим:

```
c = 24
0:24
```

Операция нахождения остатка в разных языках программирования реализуется по-разному для отрицательного делимого или делителя. В некоторых языках такая операция не определена (Basic) или остаток всегда неотрицательный, как в математике (Oberon). В других языках остаток может быть отрицательным. Возможны два варианта:

- знак остатка совпадает со знаком делимого (C++, C#, Pascal);
- знак остатка совпадает со знаком делителя (Python, Prolog).

Есть языки, в которых реализованы оба варианта (Java, Haskell). В этом случае операции обозначаются по-разному. Для обозначения операции целочисленного деления часто используют `\`, `//`, `div`, `quot` или знак, которым также обозначают деление для рациональных чисел, `/`. Для операции нахождения остатка используют `%`, `\%`, `mod`, `rem`. Могут использоваться другие обозначения.

**Пример 19.4.** Решение.

## V. Программа:

```

a = int(input('a = '))
#выделение последней цифры
a1 = a % 10
#выделение первой цифры
a2 = a // 10
b = a1 * 10 + a2
print('результат =', b)

```

## VI. Тестирование программы.

Запустите программу и введите значение  $a = 25$ .

Проверьте, результат должен быть следующим:

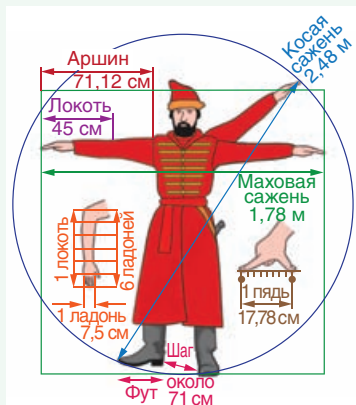
```

a = 25
результат = 52

```

На Руси традиционно применялась русская система мер. Сегодня, несмотря на то, что официально принята метрическая система мер, старорусские названия мер используются в исторических исследованиях и фразеологических оборотах.

Например: 1 аршин = 16 вершков, 1 вершок  $\approx 4,5$  см.



1) в переменной  $a1$  сохраним вторую цифру числа. Для выделения цифры из числа нужно найти остаток от деления исходного числа на 10 ( $a \% 10$ );

2) для выделения первой цифры (переменная  $a2$ ) нужно найти целую часть от деления числа на 10;

3) искомое число  $b$  получим, если умножим  $a1$  на 10 и к полученному произведению прибавим значение переменной  $a2$ .

## 3. Вывод результата.

IV. Описание переменных: все переменные имеют тип `int`.

**Пример 19.5.** В исторической книге, которую читала Таня, длина отреза ткани измерялась в локтях. Напишите программу, которая переведёт локти в метры и сантиметры.

Этапы выполнения задания.

I. Определение исходных данных: переменная  $l$  (локты).

II. Определение результатов: переменные  $m$  (метры) и  $s$  (сантиметры).

III. Алгоритм решения задачи.

1. Ввод исходных данных.

2. Сначала переведём локти в сантиметры. Для этого количество локтей нужно умножить на 45 и сохранить значение в переменной  $x$ .

3. Для определения числа метров найдём целую часть от деления  $x$  на 100.

4. Оставшиеся сантиметры можно найти как остаток от деления  $x$  на 100.

5. Вывод результата.

IV. Описание переменных: все переменные имеют тип `int`.

В Python, как и в большинстве других языков программирования, запрещено деление на нуль. Если попытаться это сделать, появится сообщение об ошибке `ZeroDivisionError`.

**Пример 19.5.** Решение.

V. Программа:

```
l = int(input('l = '))
x = l * 45
#метры
m = x // 100
#сантиметры
s = x % 100
print(l, 'локтей =', end = ' ')
print(m, 'м', s, 'см')
```

V. Тестирование программы.

Запустите программу и введите значение  $l = 7$ . Проверьте, результат должен быть следующим:

```
l = 7
7 локтей = 3 м 15 см
```



1. Какой тип данных можно использовать в Python для работы с целочисленными данными?
2. Какие операции определены для целочисленных данных?
3. Как определяется результат целочисленной операции `//` для чисел с разными знаками?
4. Как определяется результат целочисленной операции `%` для чисел с разными знаками?



## Упражнения

**1** Вася написал программу, которая переводит длину из метров в километры и метры. Но он не может решить, где нужно использовать `//`, а где — `%`. Помогите ему. Откройте файл и исправьте программу.

```
d = int(input('d = '))
k = d ... 1000
m = d ... 1000
print(d, 'м =', end = ' ')
print(k, 'км', m, 'м')
```

- 2 Ответьте на вопросы по примеру 19.4 на с. 137.
1. При каких значениях переменной  $a$  значение переменной  $b$  будет таким же?
  2. Всегда ли в результате выполнения программы получается двузначное число? Почему?
  3. Попробуйте ввести трёхзначное число (например, 125). Объясните получившийся результат.
- 3 Напишите программы для решения задач. Используйте операции // и %.
1. Задано положительное двузначное число. Найдите среднее арифметическое цифр числа.
  2. Задано положительное двузначное число. Найдите разность между количеством десятков и единиц.
  3. Дана масса в граммах. Переведите её в килограммы и граммы.
  4. Площадь участка измеряется в арах. Выразите эту площадь в квадратных километрах.
  5. Размер файла задан в килобайтах. Переведите его в мегабайты и килобайты.
- 4\* Для старорусской системы весов известны следующие соотношения:  
1 берковец = 10 пудов = 400 фунтов = 38 400 золотников.  
Напишите программу, которая переводит массу, заданную в золотниках, в фунты, пуды и берковцы.