

The Python logo, consisting of two interlocking snakes, is positioned on the left side of the slide. The snakes are stylized with thick outlines and filled with blue and yellow colors.

Что такое
Python и с чем
его едят

Содержание

[Как устроены языки программирования](#)

[Почему Python?](#)

[Типы данных](#)

[Модуль math](#)

[Print, input](#)

[Операторы](#)

[Условная инструкция](#)

[Code style](#)

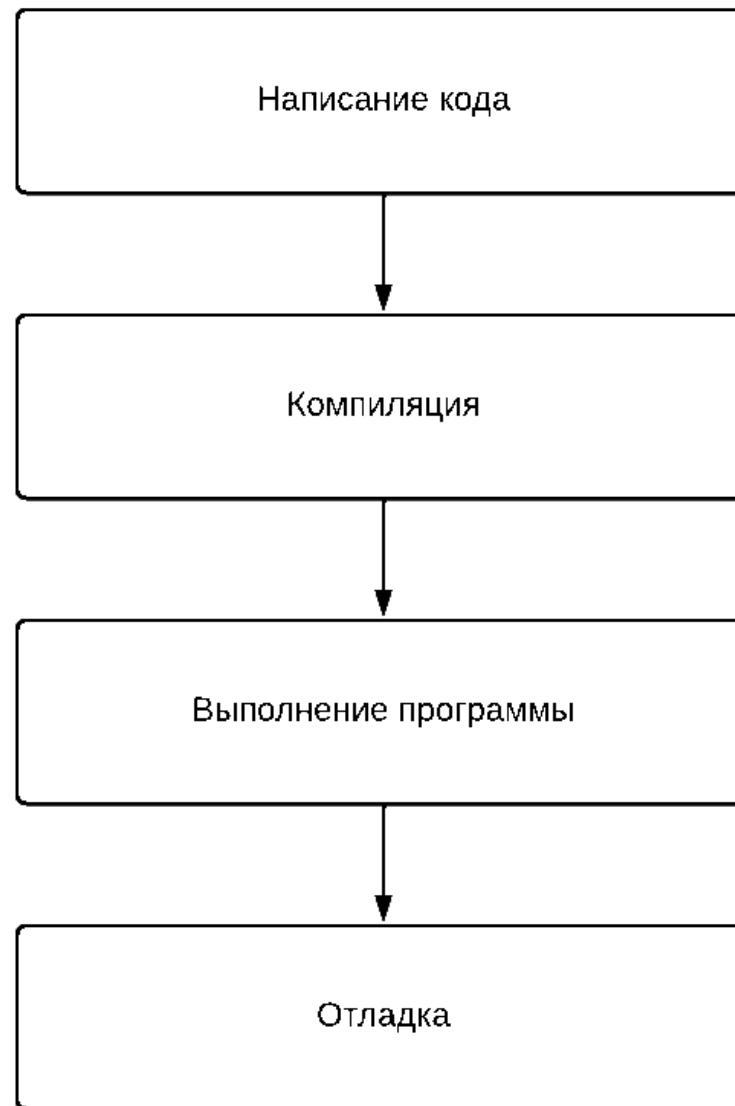
[Циклы](#)

[Функции](#)

[Работа с файлами](#)

[Справочник](#)

Этапы написания программы (сценария)



Почему именно Python?

- Выполнение программы не занимает много времени.
- Можно понять и прочитать то, что написал.
- Огромное количество поддерживаемых библиотек.
- Универсальный язык программирования.
- Используется почти во всех крупных компаниях.
- (просто посмотрите следующий слайд)

Почему именно Python?

```
#include <stdio.h>
int main(void)
{
    printf("Hello, world!");
}
```

C

```
#include <iostream.h>
int main()
{
    std::cout << "Hello, world! ";
    return 0;
}
```

C++

```
class HelloWorld {
    public static void main(String[]
args) {
        System.out.println("Hello,
World!");
    }
}
```

Java

```
print "Hello, world!"
```

Python

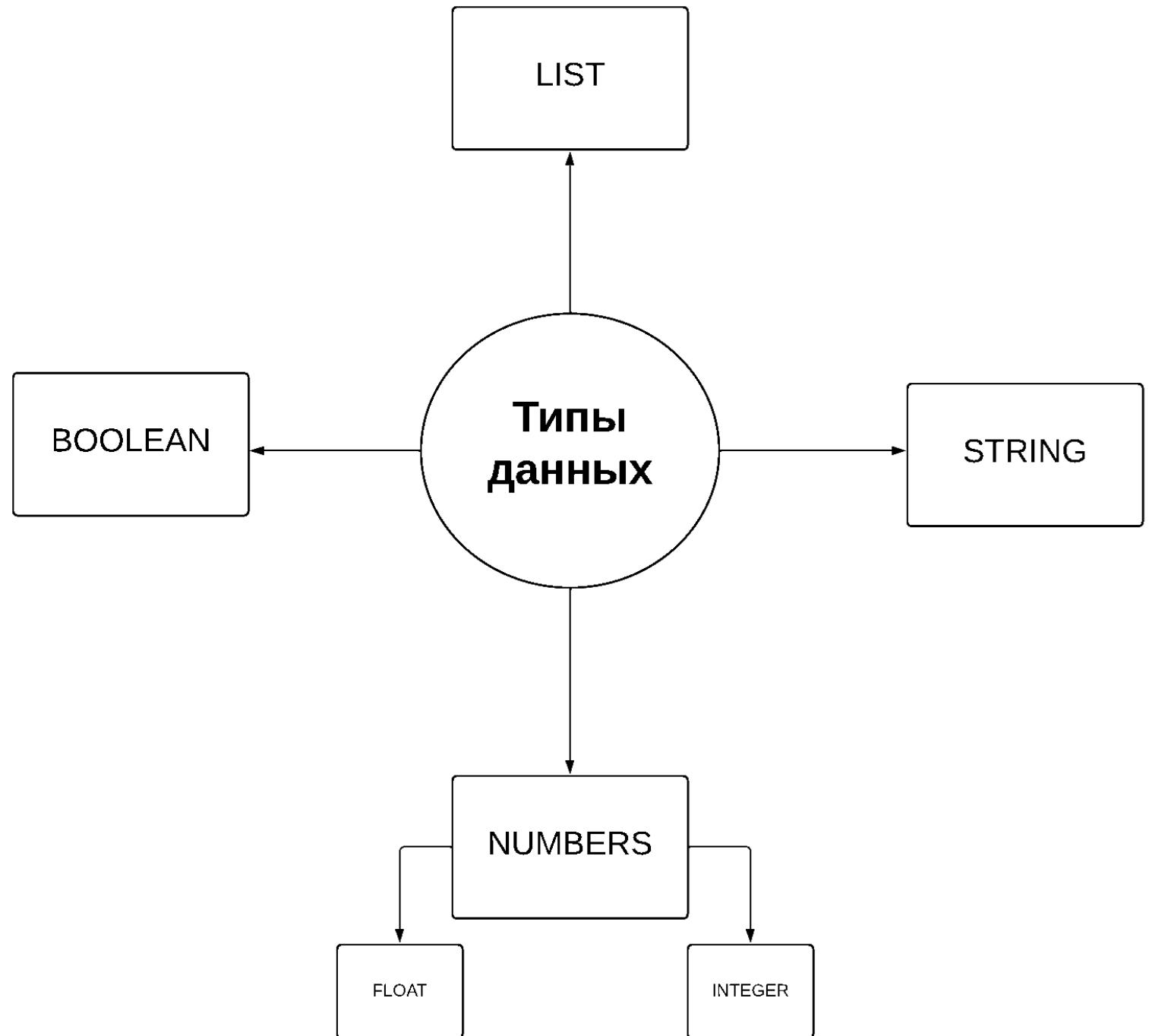


Что такое переменная?

Переменная — это хранилище данных. Сюда можно положить какое-то значение (например, число, строку или другой тип данных). Еще более простой вариант представить себе переменную — подумать о том, что нас окружает.

Типы данных

*Рассмотрим простейшие
и самые встречающиеся
типы данных*



[БОЛЕЕ ПОДРОБНО](#)

INTEGER

- Integer (int) – целочисленный тип данных. Используется для целых значений чисел (то есть те, что без дробной части):
- **Пример:** -12, -1, 0, 12, 123, 1234, 12345 и т.д.

FLOAT

- Float (float) – тип данных для дробных чисел. Если вводите целое число, то на выходе получаете десятичный формат с нулевой дробной частью. Обратите внимание, что в Python дробная и целая части разделяются точкой. При вводе запятой возникнут ошибки.
- **Пример:** 1.00, -1.23, -12.34, 12.34 и т.д.

Чем отличается `float` от `int`?

```
✓ 0 [52] a = int(5)
сек.           a
5

✓ 0 [51] b = int(5.56)
сек.           b
5

✓ 0 [39] c = float(5)
сек.           c
5.0

✓ 0 [41] d = float(5.56)
сек.           d
5.56
```

BOOLEAN

- **Логический тип данных (bool)** (или булевый тип) это примитивный тип данных, который принимает 2 значения — истина или ложь.
- В Python свой предопределённый тип `bool`, который имеет два значения (обратите внимание на написание с заглавной буквы, если напишите с маленькой компилятор выдаст вам ошибку):
 - `True` - истина
 - `False` – ложь

STRING – ОСНОВА ОСНОВ

- Стока — это упорядоченная последовательность символов, которая предназначена для хранения информации в виде простого текста.
- Разницы между строками с одинарными и двойными кавычками нет — это одно и то же
- Пример: '1234', "Hello", 'World', "-564.56,56"

MyStr

W

o

r

|

d

!

Index

0

1

2

3

4

5

Работа со строками

Конкатенация строк – `'str1' + 'str2'`

Перевод любого типа в строку – `str(element)`

Замена строки на другую – `replace(str1, str2)`

Обратиться к символу строки по индексу – `str1[index]`

Длина строки – `len(str1)`

Возвращает индекс первого вхождения подстроки в str или «-1» при отсутствии (поиск идёт в пределах от start до end) – `str1.find(str2, start, end)`

Возвращает индекс последнего вхождения подстроки в str или «-1» при отсутствии (поиск идёт в пределах от start до end) – `str1.rfind(str2, start, end)`

При работе со строками можно не только указывать единичный индекс элемента, но и создавать «срезы».

Например: `str_0 = "Hello World"`
`str_1 = str_0[:5] → str_1 == "Hello"`
`str_2 = str_0[6:11] → str_2 == "World"`

Также мы можем указывать отрицательный индекс элемента, но тогда «счёт» пойдёт с конца:

`str_3 = str_0[-1] → str_3 == "d"`
`str_4 = str_0[-5:-1] → str_4 == "worl"`

[БОЛЕЕ ПОДРОБНО](#)

[БОЛЕЕ ПОДРОБНО](#)

Работа со строками



```
[49] z = str(5) #переформатируем в строку или задаём новую строку
      z
      '5'

[81] x = 'str(5.56)' #задаем строку
      x
      'str(5.56)'

[99] str1 = 'Hello,'
      str2 = 'Hello!'
      str3 = str1+str2
      str3
      'Hello,Hello!'

[106] str3 = str3.replace("Hello", "World")
      str3
      'World,World!'
```

```
[108] str1[len(str1)-1]
      ,
      ,

[111] index1 = str3.find("World", 0, len(str3)-1)
      index1
      0

[116] index3 = str3.rfind("World", 0, len(str3)-1)
      index3
      6

[115] index3 = str3.rfind("World", 2, 5)
      index3
      -1
```

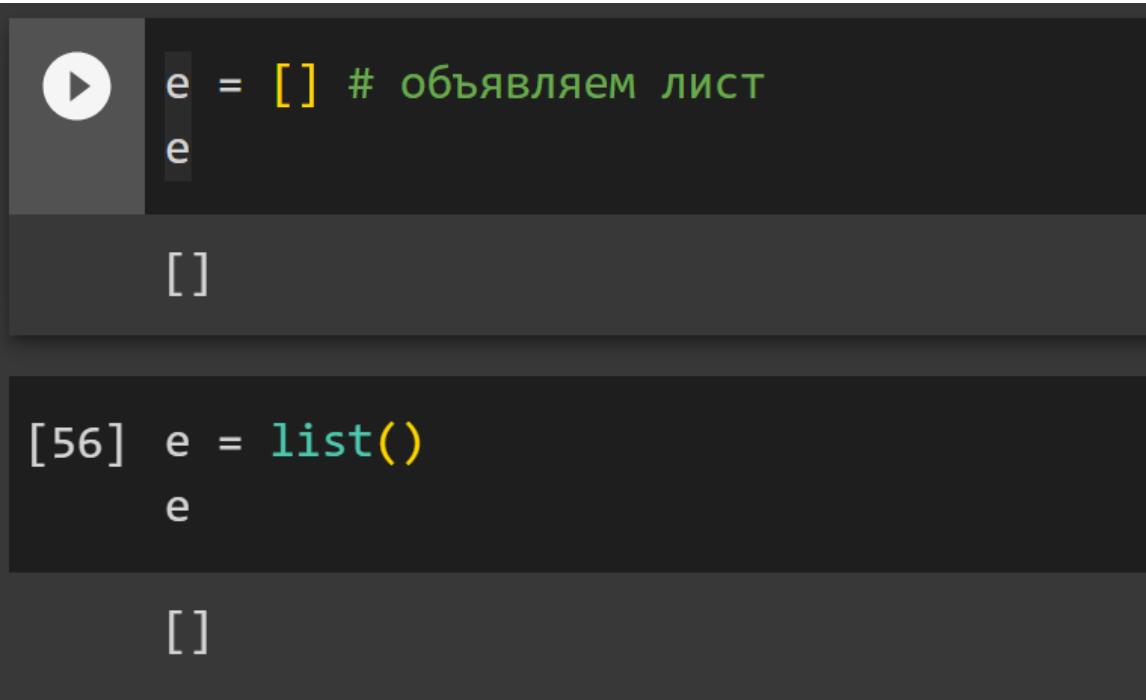
LIST

MyList	1	2	0	5	89	-9
<i>Index</i>	0	1	2	3	4	5

- List – список, тип данных, предназначенный для хранения набора или последовательности разных элементов.
- Рассмотрим как хранятся элементы в некотором списке:

Работа со списком

- Инициализировать список можно двумя способами:



```
▶ e = [] # объявляем лист
e
[]

[56] e = list()
e
[]
```

The image shows a screenshot of a Jupyter Notebook interface. A play button icon is visible on the left. The code cell contains two examples of initializing an empty list. The first example uses the syntax `e = []`, with the comment `# объявляем лист` (which translates to '# we declare a list') preceding it. The second example uses the syntax `e = list()`. Both examples result in an empty list being assigned to the variable `e`, which is then displayed as `[]`.

Работа со списком

- Мы можем заполнить лист и обращаться к любому его элементу по индексу.

```
[58] g = ['hello', 'world', '!'] #заполняем лист строками
      g
      ['hello', 'world', '!']

[59] g[0], g[1], g[2]
      ('hello', 'world', '!')
```

При работе со списками можно не только указывать единичный индекс элемента, но и создавать «срезы».

Например: myList = [1, 2 , 3, 4, 5]
myList1 = myList[:5] → myList1 == “Hello”
myList2 = myList[6:11] → myList2 == “World”

Также мы можем указывать отрицательный индекс элемента, но тогда «счёт» пойдёт с конца:

myList3 = myList[-1] → myList3 == 7
myList4 = myList[-5:-1] → myList4 == [1, 2, 3, 4]

Операции над списком

Добавление элемента (в конец списка) – `list.append(element)`

Вставка элемента в список на нужную позицию – `list.insert(index, element)`

Изменение элемента на позиции – `list[index] = element`

Удаление элемента в списке – `del list[index]`

Сортировка списка – `list.sort()`

Изменить порядок списка на противоположный – `list.reverse()`

Узнать размер списка – `len(list)`

БОЛЕЕ ПОДРОБНО

Операции над списком

```
[75] list = [1, 2, 56, 67, 78]
      list.sort()
      list
```

```
[1, 2, 56, 67, 78]
```

```
[76] list.reverse()
      list
```

```
[78, 67, 56, 2, 1]
```

```
[77] list[4] = 0
      list[2] = 1
      list
```

```
[78, 67, 1, 2, 0]
```

```
[87] list.append(23)
      list
```

```
[78, 67, 1, 2, 0, 23]
```

```
[88] list.insert(3, 44)
      list.insert(5, 66)
      list, len(list)
```

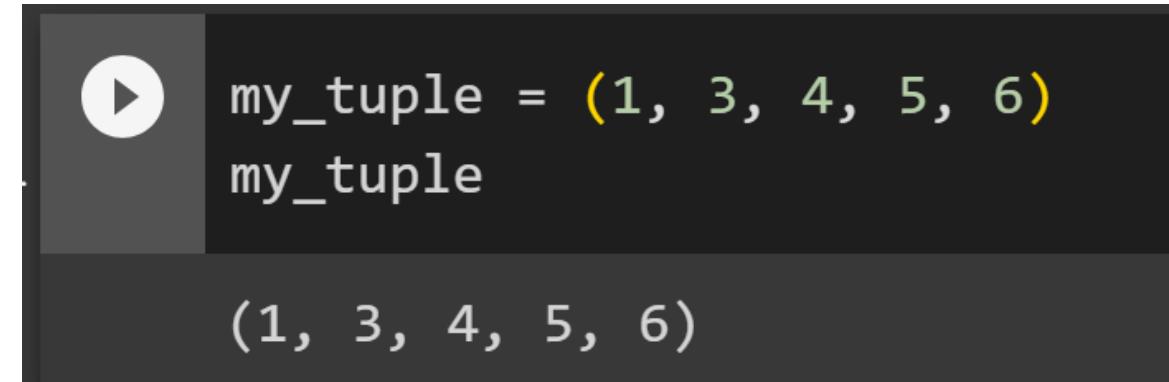
```
([78, 67, 1, 44, 2, 66, 0, 23], 8)
```

```
[89] del list[3]
      list, len(list)
```

```
([78, 67, 1, 2, 66, 0, 23], 7)
```

Кортеж

- Кортежи (tuple) в Python – это те же списки за одним исключением. Кортежи неизменяемые структуры данных. Так же как списки они могут состоять из элементов разных типов, перечисленных через запятую. Кортежи заключаются в круглые, а не квадратные скобки.
- Для кортежей используются те же методы, что и для списков.
- Чтобы объявить кортеж, мы должны воспользоваться не квадратными скобочками, как в списке, а круглыми.



The screenshot shows a code editor window with a dark theme. On the left, there is a play button icon. The main area contains the following Python code:
`my_tuple = (1, 3, 4, 5, 6)
my_tuple`
Below the code, the output is displayed in a lighter gray box:
`(1, 3, 4, 5, 6)`

Множество

Множество в python - "контейнер",
содержащий не повторяющиеся
элементы в случайном порядке.

```
my_set = {'a', 'b', 'c', 'd', 'a'}  
my_set  
{'a', 'b', 'c', 'd'}
```

Операции над множествами

`set.add(elem)` - добавляет элемент в множество.

`set.remove(elem)` - удаляет элемент из множества. `KeyError`, если такого элемента не существует.

`set.discard(elem)` - удаляет элемент, если он находится в множестве.

`set.pop()` - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.

`set.clear()` - очистка множества.

`set.update(other, ...); set |= other | ...` - объединение.

`set.intersection_update(other, ...); set &= other & ...` - пересечение.

`len(s)` - число элементов в множестве (размер множества).

`x in s` - принадлежит ли `x` множеству `s`.

БОЛЕЕ ПОДРОБНО

Словарь

Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу.

```
d = {'key1': 1, 'key2': 2}
```

```
d['key1']
```

```
1
```

Операции над словарями

- **myDictionary[keyValue]** – получить значение по ключу
- **dict.clear()** - очищает словарь.
- **dict.keys()** - возвращает ключи в словаре.
- **dict.values()** - возвращает значения в словаре.
- **dict.copy()** - возвращает копию словаря.
- **dict.clear()** - очищает словарь.

[БОЛЕЕ ПОДРОБНО](#)

Основные операторы

- **Арифметические операторы:** «+», «-», «/», «//», «%», «**», «=», «!=»
- **Логические операторы:** «and», «or», «not», «is»
- **Операторы сравнения:** «>», «<», «==», «<=», «>=»
- **Операторы принадлежности:** «in» «not in»

```
[117] 5+5  
10  
  
[118] 6-7  
-1  
  
[119] 6/4  
1.5  
  
[120] 6//4  
1
```

```
6%4  
2  
  
[121] 6*3  
18  
  
[122] 6**2  
36  
  
[128] 6<2, 6<=2, 6>3, 6>=3, 6==2, 6==6  
(False, False, True, True, False, True)
```

ФУНКЦИИ, без которых нельзя живь

- **len()** – вычисляет размер/длину объекта.
- **round()** – округляет число в большую сторону.
- **divmod()** – находит одновременно значение при делении нацело и остаток от деления.
- **sum()** – суммирует объекты.
- **min()** – нахождение минимума.
- **max()** – нахождение максимума.
- **pow()** – возведение в степень.
- **abs()** – нахождение модуля (абсолютного значения).



Встроенные функции в python



```
...
abs(), round(), divmod(), pow(), max(), min(), sum()

...
a = 5.67
a = round(a)

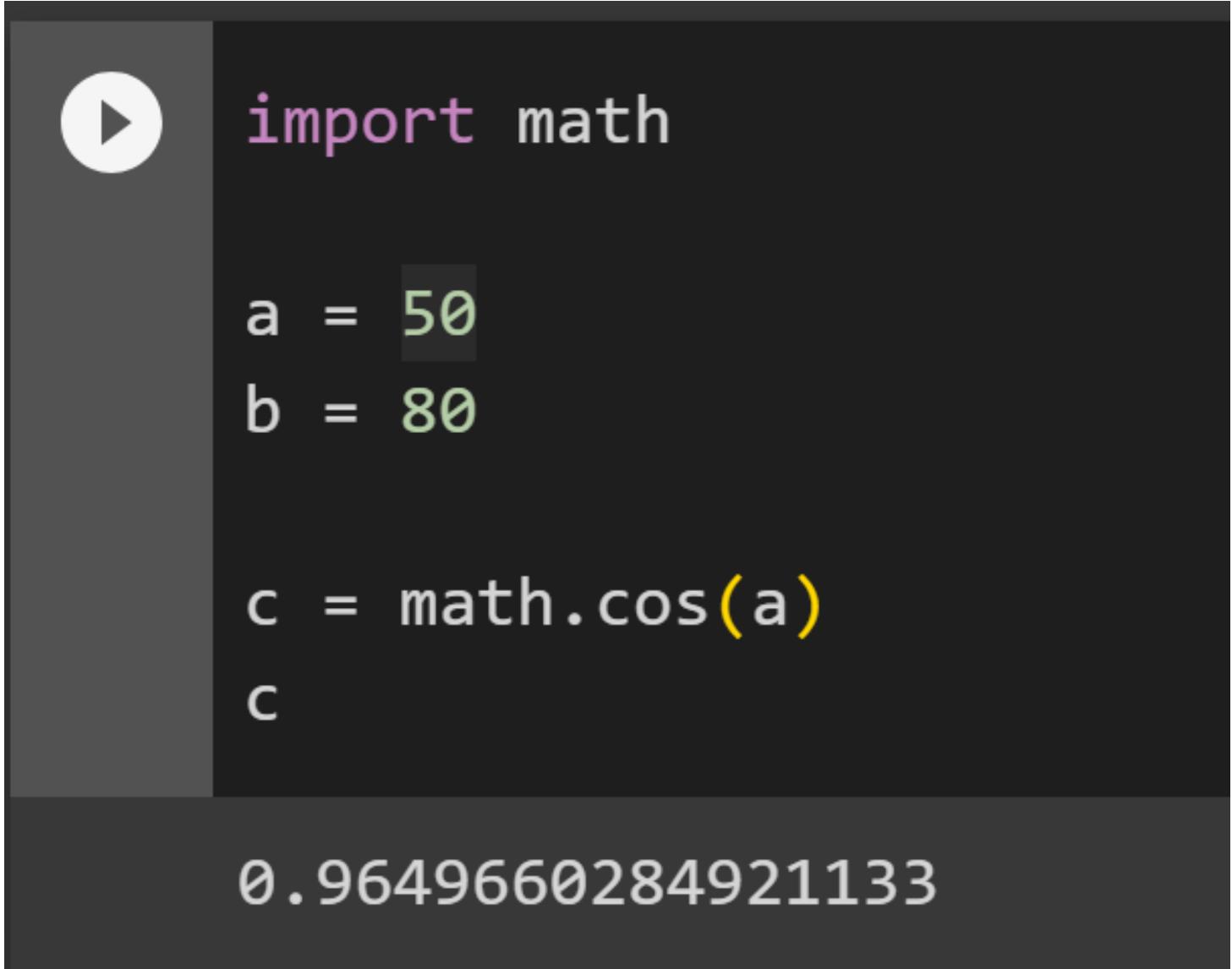
b = 6
c = 5
myList = [1, 2, 3, 4]

print(a, divmod(b, c), max(b, c), min(b, c), sum(myList), pow(b, c))

6 (1, 1) 6 5 10 7776
```

Модуль math

- Модуль math – один из наиважнейших в Python. Этот модуль предоставляет обширный функционал для работы с числами.



The image shows a Jupyter Notebook cell with the following content:

```
import math
a = 50
b = 80
c = math.cos(a)
c
```

The output of the cell is:

0.9649660284921133

Основные функции модуля `math`

- `math.fabs(X)` - модуль X.
- `math.factorial(X)` - факториал числа X.
- `math.floor(X)` - округление вниз.
- `math.log10(X)` - логарифм X по основанию 10.
- `math.pow(X, Y)` – X в степени Y.
- `math.sqrt(X)` - квадратный корень из X.
- `math.cos(X)` - косинус X (X указывается в радианах).
- `math.sin(X)` - синус X (X указывается в радианах).
- `math.tan(X)` - тангенс X (X указывается в радианах).
- `math.degrees(X)` - конвертирует радианы в градусы.
- `math.radians(X)` - конвертирует градусы в радианы.
- `math.pi` - pi = 3,1415926...
- `math.e` - e = 2,718281...

Ввод и вывод данных



Print – скорость без границ

- Функция print() используется для вывода на консоль данных.
 - Достаточно в теле функции написать, что именно вы хотите вывести (переменную, строку)
 - **Пример использования:**

А как вводить данные?

- Функция `input()` используется для ввода на консоли данных.
- Обратите внимание, что `input()` автоматически преобразует вводимые выражения в строки, поэтому нужно самостоятельно форматировать тип данных переменной.
- **Пример использования:**

```
[29] a = input("Please, input smth:")
      a

      Please, input smth:abc
      'abc'

[30] b = int(input("Please, input intenger number:"))
      b

      Please, input intenger number:5
      5

[31] c = float(input("Please, input intenger number:"))
      c

      Please, input intenger number:5
      5.0

[32] d = float(input("Please, input float number:"))
      d

      Please, input float number:5.45
      5.45

[33] e = str(input("Please, input smth:"))
      e

      Please, input smth:abc
      'abc'
```

Как ввести список?

Подробно разберем каждый шаг:

```
myList = list(map(int, input().split()))
print("Is it your list?", myList)
```

```
1 2 3 4 5
```

```
Is it your list? [1, 2, 3, 4, 5]
```

1. **input()** - функция `input()` используется для считывания строки ввода пользователя. Она ожидает, пока пользователь не введет что-то и нажмет клавишу Enter. Затем она возвращает введенную строку.
2. **split()** - метод `split()` вызывается на строке и разделяет ее на отдельные элементы (слова) по пробелам (по умолчанию). В данном случае, так как не указан разделитель, строки разделяются по пробелам. Метод `split()` возвращает список, содержащий отдельные элементы.
3. **map(int, ...)** - функция `map()` применяет функцию `int()` к каждому элементу списка. Функция `int()` преобразует значение в целое число. В данном случае, все элементы списка, полученного после разделения строки, преобразуются в целые числа.
4. **list(...)** - функция `list()` преобразует результат выполнения функции `map()` в список. Таким образом, получается список целых чисел.

Операторы. Задача №1

На вход подаются два числа: количество яблок, которое выращивает первая и вторая фермы в год. Программа должна вывести сколько яблок в год производят обе фермы вместе.

- Входные данные: 5600 4400
- Выходные данные: 10000

Операторы. Задача №2

На вход подаются два числа:
количество яблок,
выращиваемых на ферме в год,
и количество фермеров,
собирающих яблоки. Программа
должна вывести сколько яблок в
среднем собирает один фермер.

Входные данные: 4500 66
Выходные данные: 68

Операторы. Задача №3

На вход подаются три числа: количество яблок, выращиваемых на ферме в год, количество фермеров, собирающих яблоки и число яблок, приходящихся на одного фермера по мнению руководства фермы. Программа должна True, если руководство верно посчитало количество, или False, если неверно.

Входные данные: 4500 66 68

Выходные данные: True

Входные данные: 4500 60 68

Выходные данные: False

Операторы. Задача №4

На рынке покупатель спрашивает, есть ли такой сорт яблок в ассортименте. На вход подаётся список, содержащий все сорта яблок в наличии, и некоторая строка `apple_variety`. Программа должна вывести, есть ли `apple_variety` в наличии.

Входные данные:

`['Gold', 'BlackPrince', 'Caramel'], 'Gold'`

Выходные данные: `True`

Условная инструкция

if test1:

 state1



 } Тело цикла, если выполнилось
 условие test1

elif test2:

 state2



 } Тело цикла, если выполнилось
 условие test2

else:

 state3



 } Тело цикла, если значение не
 удовлетворяет ни одному из
 условий

Обратите внимание на табулирование,
ключевые слова и приоритетность
операций!

```
a = 12  
b = 34  
  
if (a + b) > 50:  
    print("Good")  
elif (a - b) < 30:  
    print("Better")  
else:  
    print("The best")
```

→ Better

```
a = 50  
b = 0  
  
if (a + b) > 50:  
    print("Good")  
elif (a - b) < 30:  
    print("Better")  
else:  
    print("The best")
```

The best

```
a = 50  
b = 34  
  
if (a + b) > 50:  
    print("Good")  
elif (a - b) < 30:  
    print("Better")  
else:  
    print("The best")
```

Good

Условная инструкция

Условная инструкция. Задача №1

На вход подаётся число.
Определить является оно
четным или нечетным.

- Входные данные: 3
- Выходные данные: ‘Число нечетное’

- Входные данные: 2
- Выходные данные: ‘Число четное’

Условная инструкция. Задача №2

Известно, что нейронная сеть наиболее точно определяет исход, если вероятность $P(A) = m/n$ события А превышает 80%. На вход подаются: n — общее число всех равновозможных, элементарных исходов этого испытания, а m — количество элементарных исходов, благоприятствующих событию А. Определить будет ли точной нейронная сеть или нет.

- Входные данные: 70 35
- Выходные данные: ‘Нет’

- Входные данные: 70 65
- Выходные данные: ‘Да’

Условная инструкция. Задача №3

Вы создаёте игру. Персонаж игрока вступает в поединок. По окончании боя в программу передаются: имя персонажа, его здоровье до поединка и количество отнятых единиц здоровья соперником. Определить выиграл персонаж или проиграл бой (то есть значение его здоровья больше нуля, либо меньше или равно).

- Входные данные: 'Sirius' 45 30
- Выходные данные: 'Игрок Sirius выиграл бой'

Условная инструкция. Задача №4

Робот Майдодыр приветствует своих хозяев разными выражениями. Если хозяин родился зимой или летом, то он говорит «Привет, хозяин!», если осенью или весной, то «Привет, дружок!». На вход подается время года рождения хозяина. Программа должна вывести верное приветствие.

- Входные данные: “Зима”
- Выходные данные: “Привет, хозяин!”

- Входные данные: “Осень”
- Выходные данные: “Привет, дружок!”

Условная инструкция. Задача №5

Яндекс разработал определитель телефонных номеров. Если номер помечается флагом «Спам», то он блокирует его и посылает автоответчику код «0». Если номер помечен флагом «Друг», он посылает автоответчику код «220», если номер помечен флагом «Важно», то отправляет ему код «44». Во всех остальных случаях он направляет код «-1». На вход подаётся флаг. Программа должна вывести код, который передаст определитель автоответчику.

- Входные данные: “Спам”
• Выходные данные: -1

- Входные данные: “Маша”
• Выходные данные: 0

Как писать комментарии в Python

```
[ ] # я  
# пишу  
# однострочные  
# комментарии  
  
[ ] """  
Это уже большой блок комментариев  
"""  
  
[ ] ...  
Это тоже большой блок комментариев  
...  
...
```

Выберете себе единственный стиль кавычек – разницы между ними нет. Используйте во всём коде только один вид!

Названия переменных

Название	Когда использовать?	Что важно?	Пример
Верблюжий стиль (camelCase)	Много слов в названии	Первое слово с маленькой буквы, остальные с заглавной, БЕЗ РАЗДЕЛИТЕЛЕЙ	myNewValue catOrDog
Змеиный стиль (snake_case)	1-3 слова в названии	Все слова пишутся с маленькой буквы, разделительный знак «_»	my_function yellow_flag

Оформление кода

Рекомендуется использовать 4 пробела на каждый уровень отступа. Python 3 запрещает смешивание табуляции и пробелов в отступах. Код, в котором используются и те, и другие типы отступов, должен быть исправлен так, чтобы отступы в нем были расставлены только с помощью пробелов.

Хорошо

```
def no_tab_using():
    no_tab = 'Using 4 spaces'
```

Плохо

```
def use_tab():
    one_tab_using = 'Ugly'
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

4. Пробелы в выражениях и инструкциях

4.1 Пробелы и скобки

4.1.1 Не ставьте пробелы внутри каких-либо скобок (обычных, фигурных и квадратных).

Хорошо

```
pineapple(pine[1], {apple: 2})
```

Плохо

```
pineapple( pine[ 1 ], { apple: 2 } )
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

4.1.2 Никаких пробелов перед открывающей скобкой, которая начинает список аргументов, индекс или срез.

Хорошо

```
get_number_of_guests(1)
```

Плохо

```
get_number_of_guests (1)
```

Хорошо

```
dish['ingredients'] = cook_book[:3]
```

Плохо

```
dish ['ingredients'] = cook_book [:3]
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

4.2 Пробелы рядом с запятой, точкой с запятой и точкой

4.2.1 Перед запятой, точкой с запятой либо точкой не должно быть никаких пробелов. Используйте пробел после запятой, точки с запятой или точки (кроме того случая, когда они находятся в конце строки).

Хорошо

```
if number_of_goods == 4:  
    print(number_of_goods, total_price)
```

Плохо

```
if number_of_goods == 4:  
    print(number_of_goods, total_price)
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

4.3 Пробелы вокруг бинарных операторов

4.3.1 Окружайте бинарные операторы одиночными пробелами с каждой стороны. Это касается присваивания (`=`), операторов сравнения (`==`, `<`, `>`, `!=`, `<>`, `<=`, `>=`, `in`, `not in`, `is`, `is not`), и булевых операторов (`and`, `or`, `not`). Используйте, как вам покажется правильным, окружение пробелами по отношению к арифметическим операторам, но расстановка пробелов по обеим сторонам бинарного оператора придает целостность коду.

Хорошо

```
counter •==• 1
```

Плохо

```
counter<1
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

4.3.3 Не используйте пробелы по сторонам знака `=`, когда вы используете его, чтобы указать на именованный аргумент или значение по умолчанию.

Хорошо

```
def complex(real, · imag=0.0): · return · magic(r=real, · i=imag)
```

Плохо

```
def complex(real, · imag · = · 0.0): · return · magic(r · = · real, · i · = · imag)
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

5. Длина строк

Ограничивайте длину строк 79 символами (а длину строк документации и комментариев — 72 символами). В общем случае не используйте обратный слеш в качестве перехода на новую строку. Используйте доступное в Python явное объединение строк посредством круглых и фигурных скобок. Если необходимо, можно добавить дополнительную пару скобок вокруг выражения.

Хорошо

```
1 | style_object(self, width, height, color='black', design=None,
2 | .....emphasis=None, highlight=0)
3 |
4 | if (width == 0 and height == 0 and
5 | .....color == 'red' and emphasis == 'strong'):
```

Если ваш текст не помещается в одну строку, используйте скобки для явного объединения строк.

Хорошо

```
long_string = ('This will build a very long long '
.....'long long long long long string')
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

7. Имена

Имена, которых следует избегать:

- Односимвольные имена, исключая счетчики либо итераторы. Никогда не используйте символы `l` (маленькая латинская буква «эль»), `O` (заглавная латинская буква «о») или `I` (заглавная латинская буква «ай») как однобуквенные идентификаторы. В некоторых шрифтах эти символы неотличимы от цифры один и нуля. Если очень нужно `l`, пишите вместо неё заглавную `L`.

Хорошо

```
long_name = 'Хорошее · имя · переменной'  
L = 'Допустимо, · но · лучше · избегать'
```

Плохо

1	<code>l = 1</code>
2	<code>I = 1</code>
3	<code>O = 0</code>

- Дефисы и подчеркивания в именах модулей и пакетов.

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Оформление кода

- Дефисы и подчеркивания в именах модулей и пакетов.

Хорошо

```
import my_module
```

Плохо

```
import my-module
```

- Двойные подчеркивания (в начале и конце имен) зарезервированы для языка.

Хорошо

```
my_variable = 'Variable'
```

Плохо

```
__myvariable__ = 'Variable'
```

Очень подробно обо всём
рассказано на этом сайте
(щёлкните по прямоугольнику)

Циклы

Представим, что нам дали ведро картошки и сказали всё его почистить.

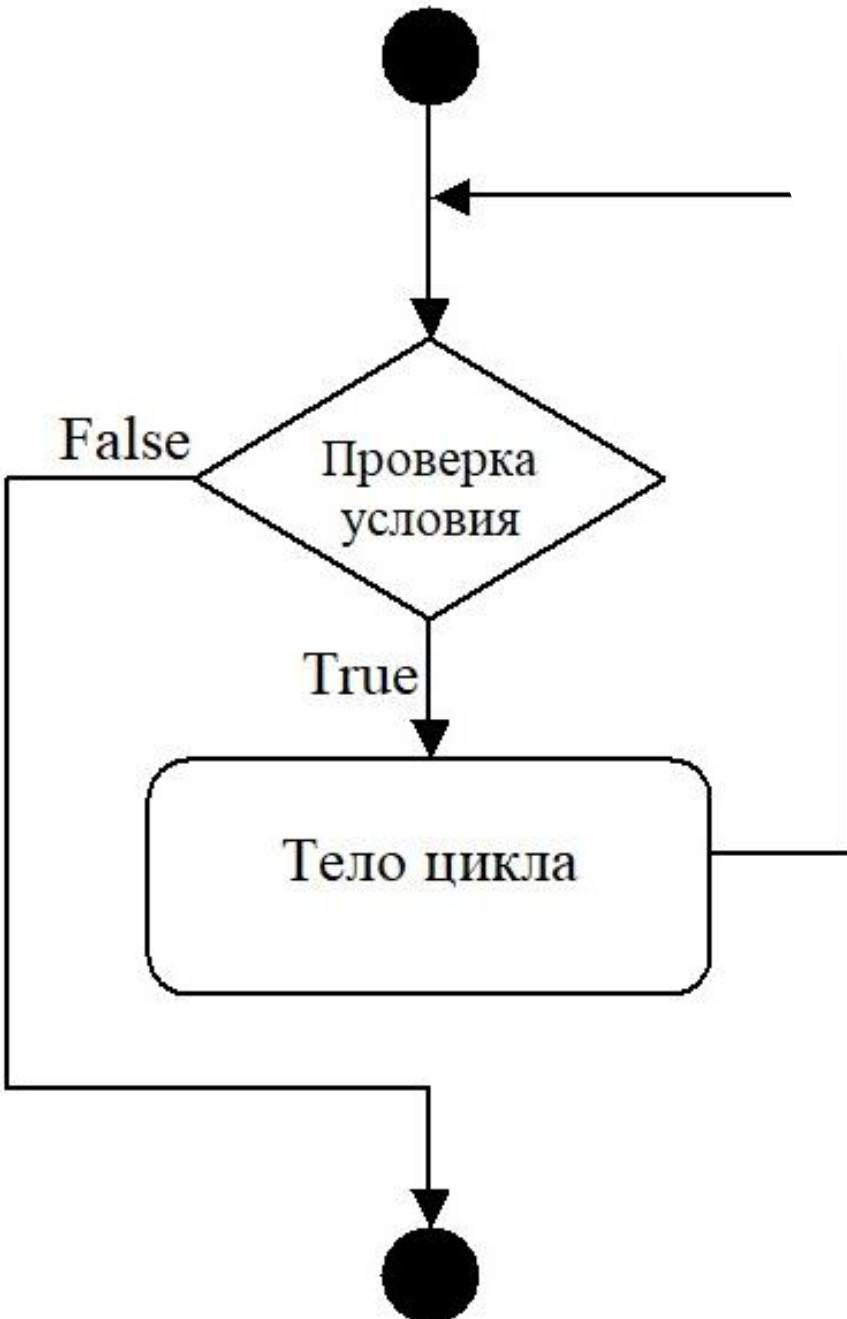
Мы знаем сколько раз мы почистим картошку?

- Нет.

А как долго мы будем выполнять это действие?

- Пока не закончится картошка.

Как решить задачу, где количество действий не известно?



Цикл while

Цикл `while` с условием в Python используется, когда точное число повторений неизвестно и может изменяться в зависимости от поведения переменной в теле цикла.

«While» можно перевести с английского как «до тех пор, пока». Этот оператор будет выполнять тело цикла до тех пор, пока выражение в его условии остаётся истинным. Чтобы условие действительно имело шанс стать ложным, в теле цикла нужно изменить переменную, которая может использоваться как счётчик итераций.

Цикл while

while условное_выражение:
инструкции

while количество_картошек != 0:
чистить картошку

Обратите внимание на табулирование,
ключевые слова и важность конечного
условия, иначе цикл уйдёт в
бесконечность!

Цикл while



n = 50
k = 0

```
while n > 6:  
    k += n  
    n -= 1
```

n, k

(6, 1254)

Циклы

Представим, что нам дали 5 картошин и сказали их почистить.

Мы знаем сколько раз мы почистим картошку?

- Да, 5 раз.

А как долго мы будем выполнять это действие?

- Пока не закончится картошка.

Как решить задачу, где количество действий известно?



Цикл for

«For» с английского переводится как «для». Этот вид цикла используют, когда количество итераций зависит от того, сколько в условии задано элементов.

Внутри цикла for нельзя изменять итерируемый объект (переменная i) — это может привести к ошибкам. Его можно использовать для вывода или параметров в промежуточных вычислениях, но новые значения лучше не присваивать. Если нужно изменить переменную, возможно, понадобится другой цикл — while.

Цикл for

for [элемент] **in** [последовательность]:
[тело цикла]

for i in range(start, end, step)

Start – начальное значение счётчика *i* (по умолчанию 1)

End – конечное значение счётчика *i*

Step – шаг изменения счётчика *i* (по умолчанию 1)

Если нужно пройтись по списку:

```
[148] mylist = [1, 2, 3, 4]  
  
for i in mylist:  
    print("my list!")
```

Если нужно пройтись по значениям счетчика:

```
[151] mylist = [1, 2, 3, 4]  
  
for i in range(0, len(mylist), 1):  
    print(mylist[i])
```

```
mylist = [1, 2, 3, 4]  
  
for i in range(len(mylist)):  
    print(mylist[i])
```

ЦИКЛЫ

- Вывод полного списка через циклы **for** и **while**

```
[19] myList = [1, 2, 3, 4, 5]
n = 0

while n != len(myList):
    print(myList[n])
    n += 1

1
2
3
4
5
```

```
[18] myList = [1, 2, 3, 4, 5]

for i in myList:
    print(i)

1
2
3
4
5
```

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

for i in range(len(myList)):
    print(myList[i])

1
2
3
4
5
```

Этот способ –
моветон!

Циклы.

Задача №1

Чтобы обеспечить безопасность данных, программисты решили каждое получаемое число преобразовывать в ближайшее, большее и противоположное им по четности число (т.е. 2 станет 3, а 5 станет 6). На вход подаётся список чисел, размер которого не ограничен. Программа должна вывести новый получившийся список.

- Входные данные: 1 4 5 89 -90
- Выходные данные: 2 5 6 90 -89

Циклы.

Задача №2

Вам на вход подаётся список чисел. Выводить этот список до первого числа, делящегося на 5. Гарантируется, что в списке обязательно содержится такое число.

- Входные данные: 1 2 6 70 8 9
- Выходные данные: 1 2 6

Циклы.

Задача №3

В банкомат разменник попадают купюры разного номинала. Определить максимальное количество пятирублёвых монет, которое может выдать разменник на данную сумму.

- Входные данные: 500
- Выходные данные: 100

- Входные данные: 363
- Выходные данные: 72

Циклы.

Задача №4

На вход подаётся некоторое число.
Определить количество его делителей
(включая единицу и само число).

- Входные данные: 7
- Выходные данные: 2
- Входные данные: 16
- Выходные данные: 5

ФУНКЦИИ

Программисты – ленивые люди.

Им очень не хочется писать много
одинакового кода. Для этого и была
придумана функция.



Что такое «функция»?

- Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции **def**.
- *Простейшая функция имеет вид:*

```
def sum(x, y):  
    return x + y
```

Обычно функции выделяют в самое начало программы

Как выглядит функция?

```
def name_of_function(arg1, arg2, ..., argN):  
    {тело функции}  
    return answer
```

Ключевое слово для обозначения функции – `def`. После него идёт имя функции, а в скобках у имени показывают принимаемые аргументы (т.е. те переменные, значения которых нам нужны в теле функции)

Возвращаем значение, которое искали в теле функции, в основной код

Тело функции содержит в себе основной её код, то есть тут мы ищем то, для чего, собственно, и писали функцию

ФУНКЦИЯ

```
def min(a, b):
    if (a < b):
        return a
    elif (a > b):
        return b
    else:
        return "Error"

n = 10
k = 10

if min(n, k) == "Error":
    print("Error")
else:
    print("min is", min(n, k))
```

```
[5] def true_or_false(year_born, today_year, age):
    if today_year - year_born == age:
        return 'True'
    else:
        return 'False'

year_born = int(input())
today_year = int(input())
age = int(input())

print(true_or_false(year_born, today_year, age))
```

```
2020
2023
5
False
```

Анонимные функции

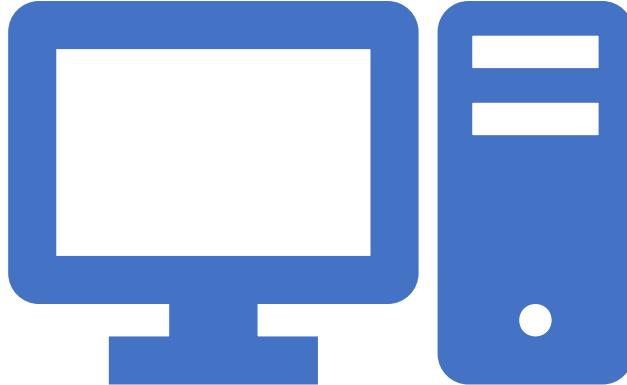
- Анонимные функции — это функции, у которых нет имени. Они определяются с помощью ключевого слова `lambda`.

```
my_function = lambda x: print(x)

x = 67
my_function(x)
```

67

Ввод и вывод данных



Файловый ввод и вывод

```
f = open("example", "r")
// работа с файлом
f.close()
```

```
with open("example", "w") as f:
    // работа с файлом
```

*Самые распространённые
режимы доступа:*

- Чтение (r)
- Добавление (a)
- Запись (w)

Файловый ввод и вывод

```
with open("example", "r") as f:  
    text = f.read()
```

```
with open("example", "r") as f:  
    part = f.read(16)
```

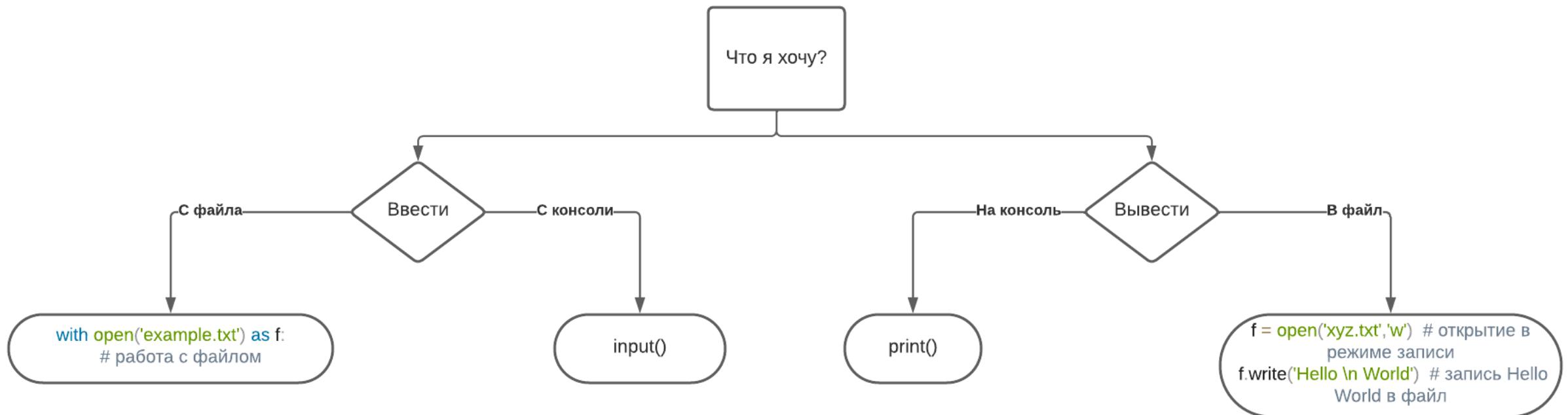
```
with open("example", "r") as f: # 'Hello, world!'  
    first_part = f.read(8)      # 'Hello, w'  
    f.seek(4)  
    second_part = f.read(8)     # 'o, world'
```

```
with open("example", "r") as f:  
    for line in f:  
        print(line)
```

```
with open("example", "w") as f:  
    f.writelines(list_of_strings)
```

```
with open("example", "w") as f:  
    print(some_data, file=f)
```

Ввод и вывод данных



ФУНКЦИЯ.

Задача №1

Смурфик Ворчун раскладывал кубики с буквами по просьбе Папы Смурфа. Но, к сожалению, он перепутал и сложил фразу из кубиков не горизонтально, а вертикально. Вам нужно помочь Ворчуна и собрать кубики в ряд. На вход принимается файл `input.txt`, в котором на каждой строчке содержится по одной букве. Пробел в файле обозначается символом «`_`». Вам нужно создать функцию, которая проверяет символы и в случае «`_`» заменяет его на пробел. Программа должна вывести в файл `output.txt` получившееся слово.

- Входные данные:

A
N
D
—
O
R

- Выходные данные: AND OR

ФУНКЦИЯ. Задача №2

В базе данных некоторой фирмы произошла ошибка: теперь у каждого сотрудника стоит неправильный дата рождения. У тех, кто родился в четный год, дата рождения увеличилась на 2 года, а у тех, кто в нечетный, наоборот, уменьшила на 6 лет. Напишите функцию, которая принимает два аргумента: год рождения и его остаток от деления на 2. На вход подаётся файл `input.txt`, в котором в каждой строке написан год рождения. После исполнения программы в файле `output.txt` должны появится новые и правильные годы.

- Входные данные:
2002
1989
- Выходные данные: 2000 1995

Функция. Задача №3

На вход подаются 2 числа: m (до 10^6) и n (число от 1 до 5). Ваша задача написать функцию, которая ищет остаток от деления числа m на 10^n . Программа должна вывести получившийся остаток.

- Входные данные: 5698 3
- Выходные данные: 698

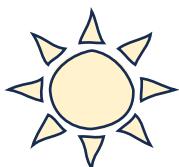
ФУНКЦИЯ. Задача №4

Дано некоторое число. Напишите функцию, которая бы помещала каждую цифру этого числа в список. Программа должна вывести сумму всех цифр числа и написать делится ли оно на 3.

- Входные данные: 123456
- Выходные данные: 21 Да
- Входные данные: 1234567
- Выходные данные: 28 Нет

Справочник

Название	Тип	Пример и краткое описание	
Integer	int	3, 79, 0, -35	- целые числа
Floating point	float	7.98, -0.905, 1.4	- упорядоченная последовательность символов
String	str	"hello", 'Ivan', "2345"	- упорядоченная последовательность символов
List	list	["Hello", 55, 26.6, "Hello"]	- упорядоченная последовательность объектов
Dictionary	dict	{"name": "Ivan", "age": 20}	- упорядоченная последовательность пар ключ-значение
Tuple	tup	("Hello", 55, 26.6, "Hello")	- упорядоченная неизменяемая последовательность объектов
Set	set	{"Hello", 55, 26.6}	- неупорядоченная коллекция уникальных объектов
Boolean	bool	True, False	- логический тип данных



Справочник

Арифметические операторы

Арифметические операторы используются для выполнения математических операций — сложения, вычитания, умножения и т. д.

Оператор	Действие	Пример
+	Сложение двух operandов или унарный плюс	$x + y + 2$
-	Вычитание правого оператора из левого или унарный минус	$x - y - 2$
*	Умножение двух operandов	$x * y$
/	Деление левого операнда на правый (результат всегда типа float)	x / y
%	Остаток от деления левого операнда на правый	$x \% y$ (остаток от x / y)
//	Деление с округлением — деление, результат которого корректируется в меньшую сторону	$x // y$
**	Показатель степени — левый operand возводится в значение правого operandана	$x^{**}y$



Справочник

Операторы сравнения

Операторы сравнения используются для сравнения значений, они возвращают `True` или `False` в зависимости от условия.

Оператор	Действие	Пример
<code>></code>	Больше чем: True, если левый операнд больше правого	<code>x > y</code>
<code><</code>	Меньше чем: True, если левый операнд меньше правого	<code>x < y</code>
<code>==</code>	Равно: True, если операнды равны между собой	<code>x == y</code>
<code>!=</code>	Не равно: True, если операнды не равны между собой	<code>x != y</code>
<code>>=</code>	Больше или равно: True, если левый операнд больше или равен правому	<code>x >= y</code>
<code><=</code>	Меньше или равно: True, если левый операнд меньше или равен правому	<code>x <= y</code>



Справочник

Логические операторы

Операторы `and` , `or` , `not` — логические.

Оператор	Действие	Пример
<code>and</code>	True, если значения обоих operandов True	<code>x and y</code>
<code>or</code>	True, если значение одного из operandов True	<code>x or y</code>
<code>not</code>	True, если значение операнда False (дополняет значение операнда)	<code>not x</code>



Справочник

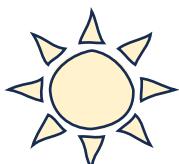
Операторы присваивания

Операторы присваивания используются для назначения переменной некоторого значения.

`a = 5` — простой оператор присваивания, который приравнивает значение 5 справа переменной `a` слева.

В Python множество составных операторов, подобных `a += 5` — он прибавляет 5 к переменной `a` и позже присваивает ей получившееся значение. Этот оператор равносителен записи `a = a + 5`.

Оператор	Пример	Эквивалентно
<code>=</code>	<code>x = 5</code>	<code>x = 5</code>
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>x -= 5</code>	<code>x = x - 5</code>
<code>*=</code>	<code>x *= 5</code>	<code>x = x * 5</code>
<code>/=</code>	<code>x /= 5</code>	<code>x = x / 5</code>
<code>%=</code>	<code>x %= 5</code>	<code>x = x % 5</code>
<code>//=</code>	<code>x //= 5</code>	<code>x = x // 5</code>
<code>**=</code>	<code>x **= 5</code>	<code>x = x ** 5</code>
<code>&=</code>	<code>x &= 5</code>	<code>x = x & 5</code>
<code> =</code>	<code>x = 5</code>	<code>x = x 5</code>
<code>^=</code>	<code>x ^= 5</code>	<code>x = x ^ 5</code>
<code>>>=</code>	<code>x >>= 5</code>	<code>x = x >> 5</code>
<code><<=</code>	<code>x <<= 5</code>	<code>x = x << 5</code>

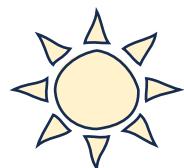


Справочник

Операторы тождественности

`is` и `is not` — операторы тождественности в Python. Они проверяют, находятся ли два значения (или две переменные) по одному адресу в памяти. То, что две переменные равны еще не значит, что они идентичны.

Оператор	Действие	Пример
<code>is</code>	True, если operandы идентичны (указывают на один объект)	<code>x is True</code>
<code>is not</code>	True, если operandы не идентичны (не указывают на один объект)	<code>x is not True</code>

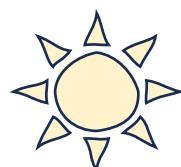


Справочник

Операторы принадлежности

`in` и `not in` — операторы принадлежности в Python. Они проверяют, есть ли значение или переменная в последовательности (строке, списке, кортеже, множестве или словаре). Иначе говоря, проверяют вхождение элемента в коллекцию. В словаре можно проверить только присутствие ключа, не значения.

Оператор	Действие	Пример
<code>in</code>	True, если значение или переменная есть в последовательности	<code>5 in x</code>
<code>not in</code>	True, если значения или переменной нет в последовательности	<code>5 not in x</code>



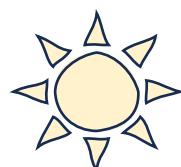
Справочник

Функции для работы со строками

Для работы со строками в Питоне предусмотрены специальные функции. Рассмотрим их:

Преобразование числового или другого типа к строке:

- `str(n)` — преобразование числового или другого типа к строке;
- `len(s)` — длина строки;
- `chr(s)` — получение символа по его коду ASCII;
- `ord(s)` — получение кода ASCII по символу.



Справочник

Методы для работы со строками

Кроме функций, для работы со строками есть немало методов:

- `find(s, start, end)` — возвращает индекс первого вхождения подстроки `s` или `-1` при отсутствии. Поиск идет в границах от `start` до `end`;
- `rfind(s, start, end)` — аналогично, но возвращает индекс последнего вхождения;
- `replace(s, new)` — меняет последовательность символов `s` на новую подстроку `new`;
- `split(x)` — разбивает строку на подстроки при помощи выбранного разделителя `x`;
- `join(x)` — соединяет строки в одну при помощи выбранного разделителя `x`;
- `strip(s)` — убирает пробелы с обеих сторон;
- `lstrip(s), rstrip(s)` — убирает пробелы только слева или справа;
- `lower()` — перевод всех символов в нижний регистр;
- `upper()` — перевод всех символов в верхний регистр;
- `capitalize()` — перевод первой буквы в верхний регистр, остальных — в нижний.



Справочник

С множествами можно выполнять множество операций: находить объединение, пересечение...

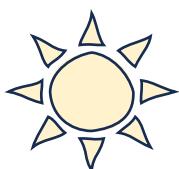
- `len(s)` - число элементов в множестве (размер множества).
- `x in s` - принадлежит ли `x` множеству `s`.
- `set.isdisjoint(other)` - истина, если `set` и `other` не имеют общих элементов.
- `set == other` - все элементы `set` принадлежат `other`, все элементы `other` принадлежат `set`.
- `set.issubset(other)` или `set <= other` - все элементы `set` принадлежат `other`.
- `set.issuperset(other)` или `set >= other` - аналогично.
- `set.union(other, ...)` или `set | other | ...` - объединение нескольких множеств.
- `set.intersection(other, ...)` или `set & other & ...` - пересечение.
- `set.difference(other, ...)` или `set - other - ...` - множество из всех элементов `set`, не принадлежащие ни одному из `other`.
- `set.symmetric_difference(other); set ^ other` - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
- `set.copy()` - копия множества.



Справочник

И операции, непосредственно изменяющие множество:

- **set.update(other, ...); set |= other | ...** - объединение.
- **set.intersection_update(other, ...); set &= other & ...** - пересечение.
- **set.difference_update(other, ...); set -= other | ...** - вычитание.
- **set.symmetric_difference_update(other); set ^= other** - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
- **set.add(elem)** - добавляет элемент в множество.
- **set.remove(elem)** - удаляет элемент из множества. KeyError, если такого элемента не существует.
- **set.discard(elem)** - удаляет элемент, если он находится в множестве.
- **set.pop()** - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.
- **set.clear()** - очистка множества.



Справочник

Методы списков

- `list.append(x)` — позволяет добавлять элемент в конец списка;
- `list1.extend(list2)` — предназначен для сложения списков;
- `list.insert(i, x)` — служит для добавления элемента на указанную позицию (`i` — позиция, `x` — элемент);
- `list.remove(x)` — удаляет элемент из списка (только первое вхождение);
- `list.clear()` — предназначен для удаления всех элементов (после этой операции список становится пустым `[]`);
- `list.copy()` — служит для копирования списков.
- `list.count(x)` — посчитает количество элементов `x` в списке;
- `list.index(x)` — вернет позицию первого найденного элемента `x` в списке;
- `list.pop(i)` — удалит элемент из позиции `i` ;
- `list.reverse()` — меняет порядок элементов в списке на противоположный;
- `list.sort()` — сортирует список.



Справочник

Методы словарей

dict.clear() - очищает словарь.

dict.copy() - возвращает копию словаря.

classmethod **dict.fromkeys(seq[, value])** - создает словарь с ключами из seq и значением value (по умолчанию None).

dict.get(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).

dict.items() - возвращает пары (ключ, значение).

dict.keys() - возвращает ключи в словаре.

dict.pop(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).

dict.popitem() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.

dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).

dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).

dict.values() - возвращает значения в словаре.

