Что такое функция

В математике

— зависимость одной величины (значения функции) от другой (аргумента функции).

$f(x) = 1 * 2^x$

- f(1) = 2
- f(3) = 8
- f(0) = ?

Что такое функция

В Python

— фрагмент кода, который можно вызвать по имени.

Функции можно **передать аргументы**. Она может **вернуть значение**.

Что такое функция

В Python

```
01. result = max(5, 3, 4)
02. print(result)
```

- Сколько тут функций?
- Какие из них принимают аргументы?
- Какие возвращают значение?

Переменная

— названная определенным именем область памяти, содержащая какое-либо значение. В переменную можно записать значение, а можно — достать его, написав её имя.

```
01. result = max(5, 3, 4)
02. print(result)
```

- Где здесь переменная?
- Где мы записываем значение, а где достаем его?

Типы данных

Внутрь переменной может быть записана информация разного типа.

```
01. age = 16
02. message = "Happy " + str(age) + "th birthday!"
```

- Сколько здесь переменных?
- Информация какого типа записывается в каждую из них?

Типы данных

- **Число** — без комментариев.
- **Строка** — последовательность символов. В Python указывается в кавычках (чтобы не путать с командами и другими типами).

Приведение типов

Данные можно преобразовывать из одного типа в другой. Для этого в Python есть специальные функции

```
01. number = int("645")
02. string = str(287)
```

- Что во что превращают эти функции?
- Что передается им в качестве аргументов и что они возвращают?

Зачем приводить типы?

Разные типы данных ведут себя по-разному

```
01. print(5 + 67 + 7)
02. print("5" + "67" + "7")
03. print("iam" + "who" + "ami")
```

- Что выведет функция `print` в каждом случае?

Задача

Считать от пользователя строку, состоящую из трех чисел, разделенных пробелами. Сохранить каждое число в отдельную переменную.

Ввод:

```
"10 4 81"
```

Вывод:

```
10
4
81
```

Функция input

Считывает строку, введенную пользователем, и возвращает ее.

```
string = input()
```

- Данные какого типа запишутся в переменную `string`?
- Какой аргумент принимает функция `input`?

Функция split

Делит строки на кусочки по определенному разделителю.

```
01. pieces1 = "10 20 78".split(" ")
02. pieces2 = input().split("/")
```

- Кусочки каких строк окажутся в переменных `pieces1` и `pieces2`?
- Какой аргумент принимает функция `split`?
- Можно ли не передавать никакого аргумента функции `split`?

Методы

Обратите внимание, как вызывается фунция split.

```
pieces = "10 20 78".split()
```

```
01. string = "10 20 78"
02. pieces = string.split()
```

Она применяется к конкретной строке или к переменной, содержащей строку. Такие функции называются **методами**.

Функция split

```
01. pieces = "10 20 78".split(" ")
02. print(pieces)
```

- Что возвращает фунцкия `split`?

Списки

Особый тип данных, представляющий собой перечисление строк, чисел или чего-нибудь еще.

```
01. pieces = "10 20 78".split(" ")
02. print(pieces)
```

Вывод выглядит так: `["10", "20", "78"]`

Функция map

Применяет какую-нибудь функцию ко всем элементам списка. Возвращает новый список из результатов работы этой фунции.

```
01. strings = ["10", "35", "42"]
02. numbers = map(int, strings)
```

- Что будет записано в переменную `numbers`?
- Какие аргументы принимает функция `map`?

Распаковка

Запись каждого элемента списка в отдельную переменную.

```
01. numbers = [10, 35, 42]
02. a, b, c = numbers
```

- Что будет записано в переменные `a`, `b` и `c`?

Возвращаемся к задаче

```
01. # Считываем строку, введенную пользователем "3 453 23"
02. string = input()
03. # Разбиваем строку на список строк ["3", "453", "23"]
04. pieces = string.split()
05. # Превращаем список строк в список чисел [3, 453, 23]
06. numbers = map(int, pieces)
07. # Распаковываем список в отдельные переменные
08. a, b, c = numbers
```

Возвращаемся к задаче

Все эти четыре действия можно записать в одну строку без дополнительных переменных.

```
a, b, c = map(int, input().split())
```

Как собрать нужную строку для вывода

Можно ее склеить с помощью оператора `+`.

```
01. a, b, c = 5, 89, 45
02. sum = a + b + c
03. print(str(a) + "+" + str(b) + "+" + str(c) + "=" + str(sum))
```

- Зачем здесь используется функция `str`?
- Чем `+` в кавычках отличается от `+` без кавычек?

Как собрать нужную строку для вывода

...но лучше воспользоваться особенностями функции `print`

```
01. a, b, c = 5, 89, 45
02. sum = a + b + c
03. print(a, "+", b, "+", c, "=", sum)
```

- Через какой разделитель выведутся переданные строки?

Именованные аргументы функции `print`

У некоторых функций есть специальные **именованные** аргументы.

```
01. print("hello", "10b", sep="=^__^=")
02. print("how", "doing", end="?")
```

- Какие именованные аргументы есть у функции `print`?
- Какие значения по-умолчанию у этих аргументов?

Функция round

Округляет число до нужной точности.

```
01. number = 3.14159265359
02. rounded = round(number, 2)
```

- Какие аргументы принимает функция `round`?
- Какое значение окажется в переменной `rounded`?

А что если пользователь вводит дробное число?

```
01. n1 = int(input()) # для целых чисел
02. n2 = float(input()) # для дробных чисел
```

Пи? Квадратный корень?

Библиотека `math`.

```
01. import math
02. res = math.sqrt(math.pi)
```

```
01. from math import sqrt, pi
02. res = sqrt(pi)
```

pythonworld.ru/moduli/modul-math.html

Метод `format`

Самый крутой способ собирать строку на лету.

```
print("Hello, my dear {}.".format("friend"))
```

А еще можно подставлять числа и указывать нужную точность.

```
print("Hypotenuse is {:.3f}".format(sqrt(a**2 + b**2)))
```

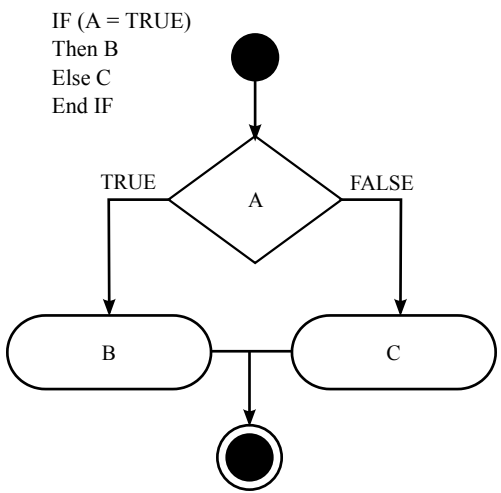
Случайные числа?

Библиотека `random`!

pythonworld.ru/moduli/modul-random.html

Ветвление

Выбор одной или другой ветки алгоритма, в зависимости от истинности логического выражения.



Логическое выражение

Простейшим логическим вырежением в Python является математическое сравнение.

```
01. age >= 18
02. length < 25
03. n**2 == 81
```

Результат логического выражения — значение **логического типа** (`True` или `False`).

Операторы сравнения в Python

- `>` – больше
- `>=` – больше или равно
- `<` – меньше
- `<=` – меньше или равно
- `==` – равно
- `!=` – не равно

Сложные логические выражения

— склыдываются из простых при помощи логических операторов.

Логические операторы

- `age >= 18 and age <= 30`
- `length == 10 or width > 15`
- `not temperature < 15`

Логический оператор `and`

— дает правду только когда оба простых выражения правдивы.

`a`	`b`	`a and b`
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

Логический оператор `or`

— дает правду когда хотя бы одно из простых выражений правдиво.

`a`	`b`	`a or b`
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`True`

Логический оператор `not`

— превращает ложь в правду, а правду — в ложь.

`a`	`not a`
`False`	`True`
`True`	`False`

Простейшее ветвление в Python

```
01. if age >= 16 and grade >= 10:
02.     print("Бегите, глупцы!")
```

Полное ветвление

```
01. if age >= 16 or grade >= 10:
02.     python_torture()
03. else:
04.     give_icecream()
05.     pat_on_the_head()
```

Каскадное ветвление

```
01. if temperature >= 20:
02.     print("Пойдем загорать на солнышке")
03. elif temperature >= 10:
04.     print("Кажется, жить ещё можно")
05. elif temperature >= 0:
06.     print("Холодненько")
07. else:
08.     print("Все. Капец. Зима")
```

Тернарный оператор

Отличный способ записать в одну строчку простое ветвление

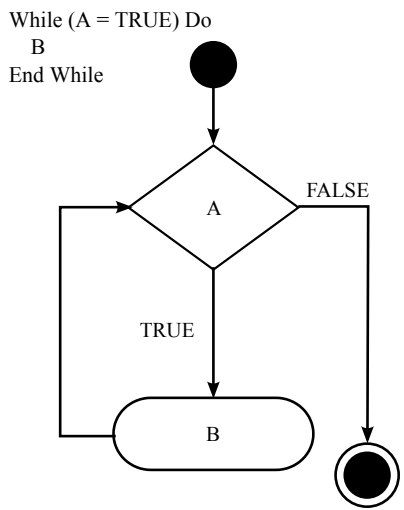
```
print("Adult" if age > 18 else "Child")
```

Решение задачи L (Точка 3)

```
01. x, y = map(float, input().split())
02. area1 = y < 2 - x**2 and y > x
03. area2 = y < 2 - x**2 and y > 0
04. print("YES" if area1 or area2 else "NO")
```

Цикл с условием

Блок кода, который выполняется снова и снова, пока условие, указанное в его начале истинно.



Простейший цикл с условием в Python

```
01. i = 0
02. while i < 100:  # заголовок цикла
03.     print(i)  # тело цикла
```

- Какие числа выведутся на экран?
- Сколько раз выполнится тело цикла?

Простейший цикл с условием в Python

Нельзя забывать в теле цикла изменять переменную, от которой зависит истинность условия в заголовке. Иначе цикл будет бесконечным.

```
01. i = 0
02. while i < 100:
03.     print(i)
04.     i += 1
```

Операции сокращенного присваивания

```
01. a += b  # a = a + b
02. a -= b  # a = a - b
03. a *= b  # a = a * b
04. a /= b  # a = a / b
05. a //= b  # a = a // b
06. a %= b  # a = a % b
```

Цикл `for`

Цикл `for` в Python перебирает все значения в каком-либо списке. В переменную, объявленную в заголовке цикла по-очереди записывается каждый из пунктов списка.

Цикл `for`

```
01. numbers = [3, 4, 5, 8, 5, 9]
02. for n in numbers:
03.     print(n, end=' ')
```

- Какие числа выведутся на экран?
- Сколько раз выполнится тело цикла?

Функция `range`

Генерирует последовательность по заданным правилам.

```
01. print(*range(10)) # 0 1 2 3 4 5 6 7 8 9
02. print(*range(5, 10)) # 5 6 7 8 9
03. print(*range(1, 10, 3)) # 1 4 7
04. print(*range(8, 2, -2)) # 8 6 4
```

Функция `range`

Генерирует последовательность по заданным правилам.

```
range(
    левая граница - включительно,
    правая граница - не включительно,
    шаг - не обязательно
)
```

Функция `range`

Очень удобно используется в связке с циклом `for`.

```
01. for i in range(4, 12, 4):
02.     print(i, end=' ')
```

- Какие числа выведутся на экран?
- Сколько раз выполнится тело цикла?