PRODUCT
STAR.RU

Функции

Содержание урока

- Что такое функции и как они используются
- Как объявить функцию в Python
- Как использовать функции
- Области видимости переменных и функции
- Множественные аргументы и аргументы по умолчанию
- Лямбда-функции
- Заключение

ИВАН МИЛОХИН

Quantitative Analyst @ Barclays Investment Bank (London, UK)

- Применял Python и C++ к задачам повышения разрешения изображений и обработки видео
- Разрабатываю библиотеки для прайсинга финансовых инструментов

Что такое функции и как они используются

Что такое функция?

Функция – это фрагмент кода, выделенный в отдельный блок.

Понятие функции пришло в программирование из математики, и чтобы его прочувствовать, давайте вспомним простейший пример математической функции:

y(x) = 5 * x + 2

Что такое функция?

Итак, мы можем передавать в функцию разные входные значения **x**, и будем получать обратно из функции некоторое новое значение, вычисленное по формуле:

y(x) = 5 * x + 2

- ★ если мы передадим в функцию **x = 1**, то наша функция в ответ вернет значение **7**, потому что **5 * 1 + 2 = 7**
- ★ если мы передадим в функцию **x = 0**, то наша функция в ответ вернет значение **2**

Тот же принцип и в программировании.

Функции в программировании

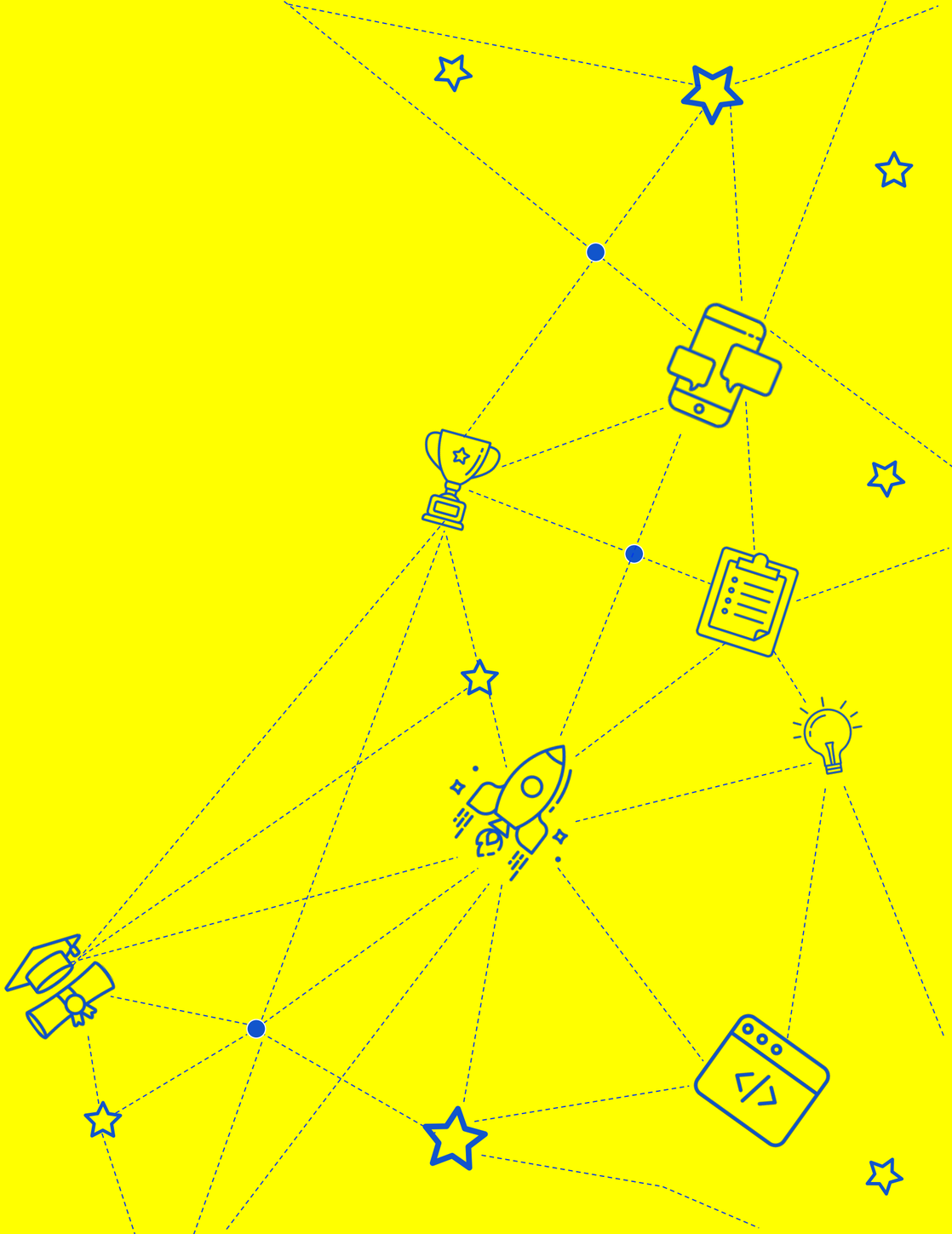
Функции в программировании, в отличии от математики, могут принимать на вход не только **числа**, но и,к примеру **строки** или **не иметь параметров.**

Давайте вспомним, какие мы знаем типы данных, которые могли бы быть аргументами функций:

- ★ целые числа
- ★ вещественные числа
- ★ строки

Какие ещё?

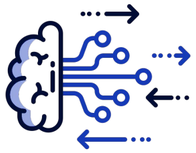
Как объявить функцию в Python



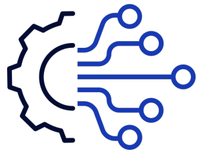
Как объявить функцию в Python

```
def <имя функции>(<аргументы>):
    <тело функции>
```

имя функции
требуется для того, чтобы мы могли к ней позже обратиться в коде, то есть вызвать

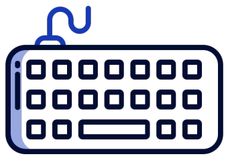


аргументы
входные данные, в зависимости от которых может меняться внутренняя логика функции и возвращаемый результат



тело функции
та логика, или же действия, которые выполняет функция

Простейшая функция



Создадим нашу первую функцию – она будет настолько простой, насколько вообще возможно!

```
def function():
  """Эта функция ничего не делает"""
  pass
```

Это действительно самая простая функция, которую только можно создать

y(x) = 5 * x + 2 на Python

Реализуем теперь математическую функцию, которую обсуждали в предыдущем разделе:

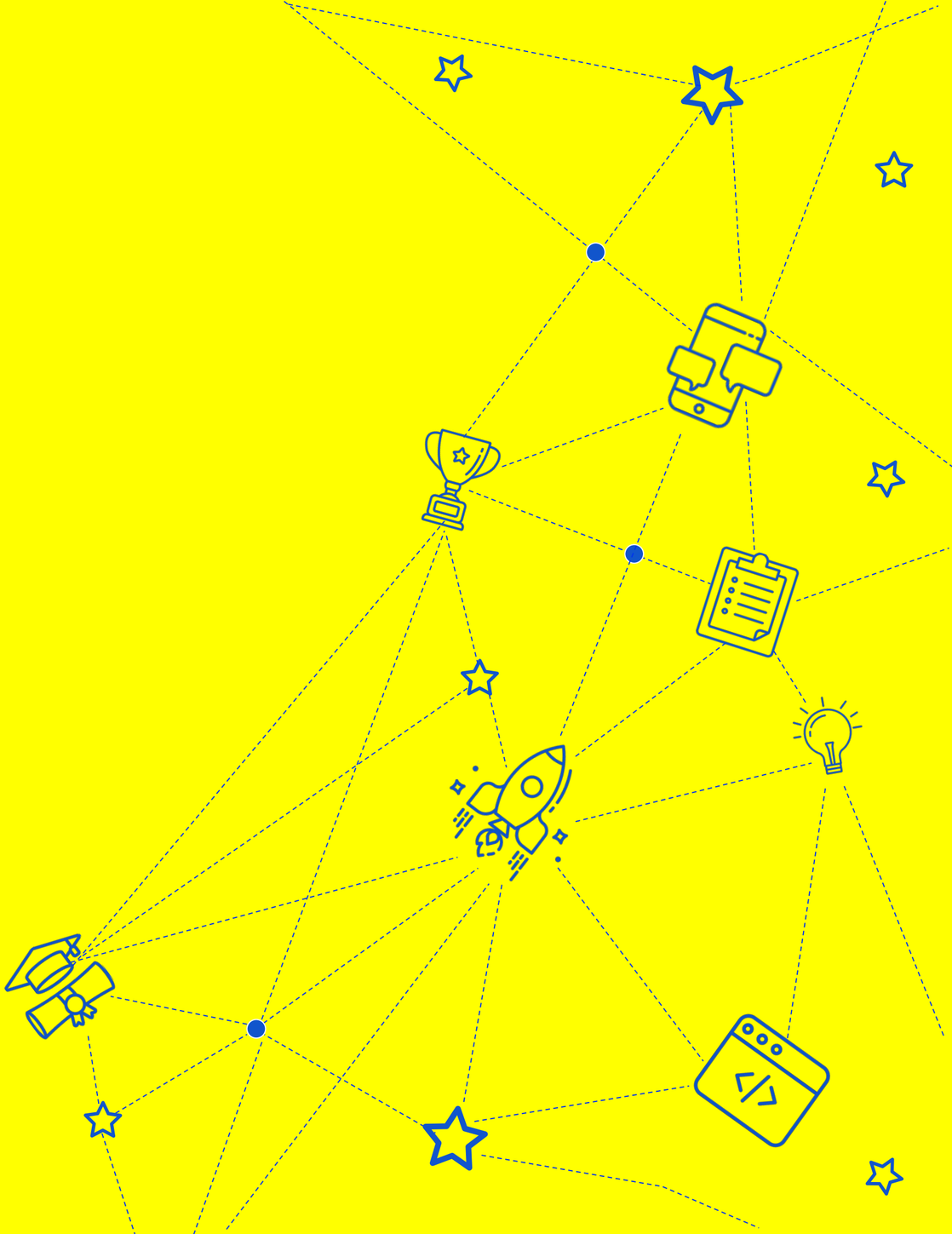
```
def y(x):
    return 5 * x + 2
```

Hello, world!
и использование библиотечных функций

```
def hello(name):
    print(f'Hello, {name}!')
```

Как же нам нам использовать нашу функцию, чтобы вывести на печать **‘Hello, world!’**?

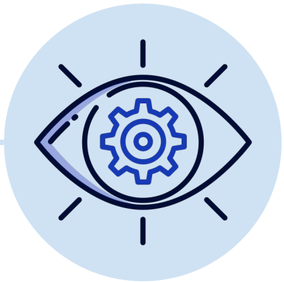
Как использовать функции



Вызов функции

Вернемся к нашему предыдущему примеру:

```
def hello(name):
    print(f'Hello, {name}!')


hello('World')
```



Если мы запустим наш пример
на выполнение, то увидим на консоли:
Hello, world!

Использование библиотечных функций

```
>>> help(print)
Help on built-in function print in module builtins:
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Добавляем справку

```
def hello(name):
    """Функция, которая печатает на консоль приветствие"""
    print(f'Hello, {name}!')


hello('World')
```

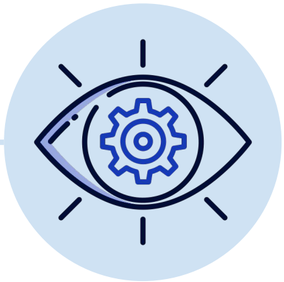
Теперь, если мы попробуем получить справку для нашей функции, мы увидим нечто подобное:

```
>>> help(hello)
Help on function hello in module main:
```

hello(name) функция, которая печатает на консоль приветствие

А что там с y(x) = 5 * x + 2?

```
def y(x):
    return 5 * x + 2


print(y(0))
print(y(1))
print(y(10))
```



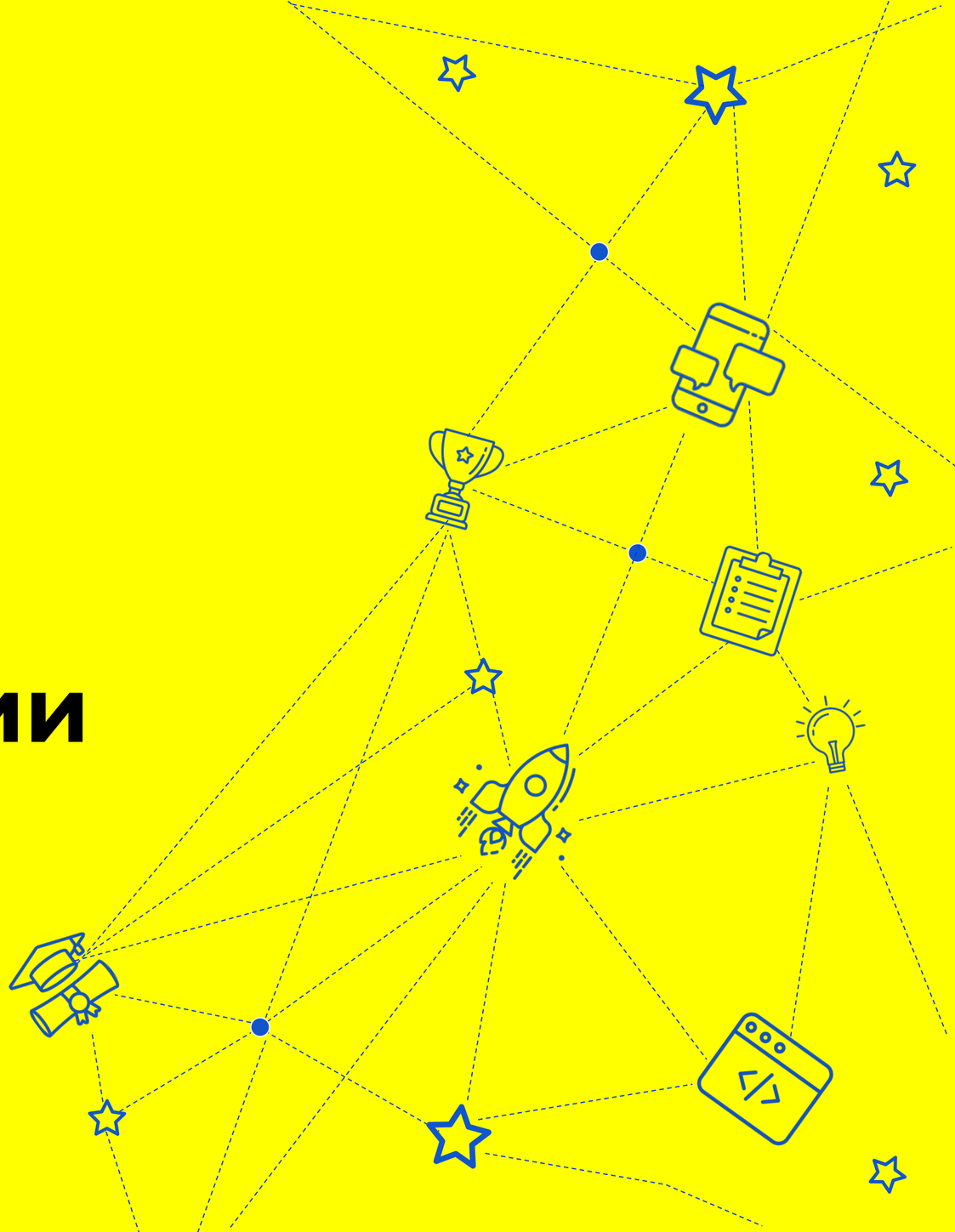
На экране мы увидим:

2

7

52

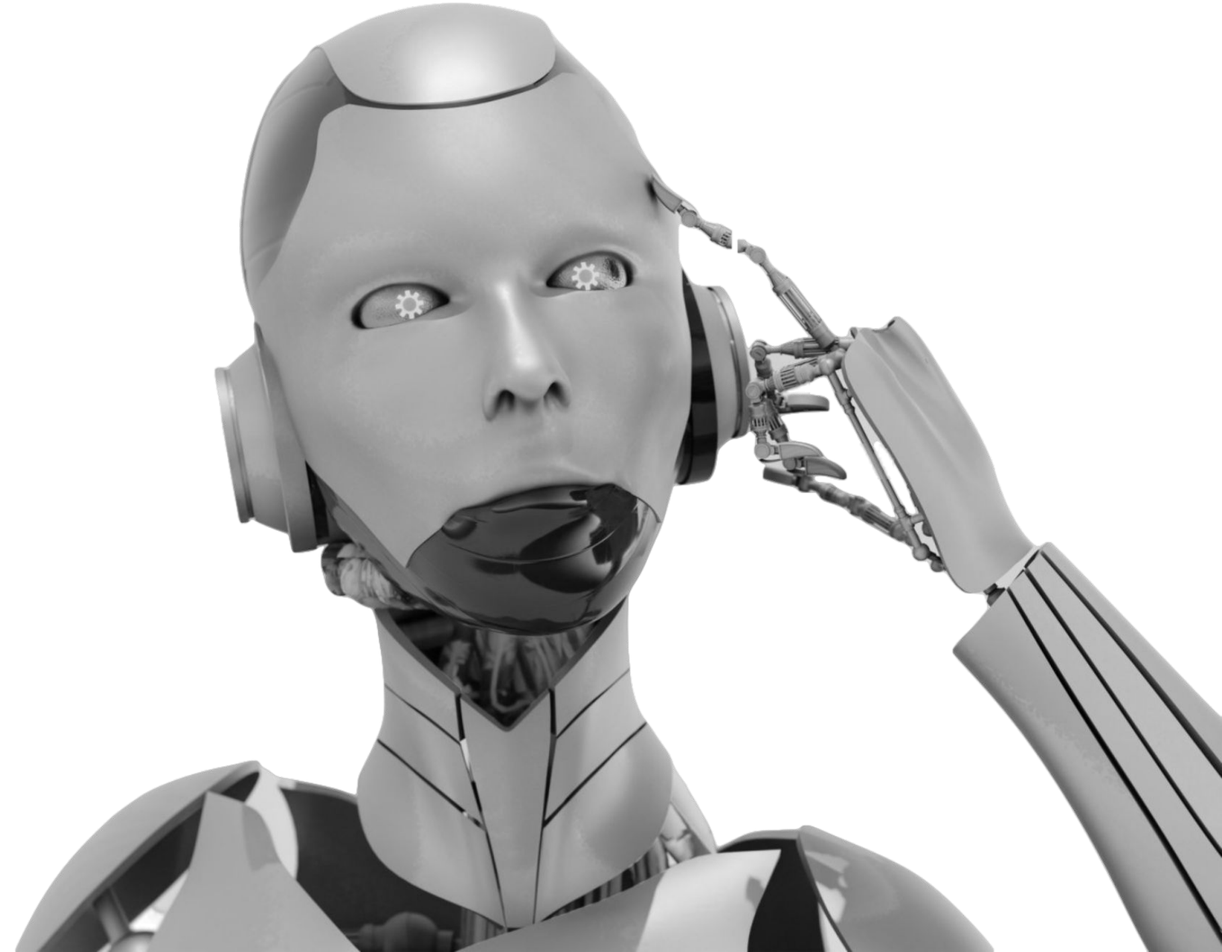
Области видимости переменных и функции



Использование глобальных переменных

a = 5
b = 2

```
def y(x):
    return a * x + b


print(y(0))
print(y(1))
print(y(10))
```



А что если сделать так?

Какое значение **a** будет использовано?

a = 5
b = 2

```
def y(x):
    a = 10
    return a * x + b


print(f'y(1) = {y(1)}')
print(f'a = {a}')
```

А что если сделать так?

Какое значение **a** будет использовано?

a = 5
b = 2

```
def y(x):
    a = 10
    return a * x + b


print(f'y(1) = {y(1)}')
print(f'a = {a}')
```

```
# y(1) = 12
  a = 5
```

Как менять глобальные переменные из тела функции

a = 5
b = 2

```
def y(x):
    global a
    a = 10
    return a * x + b


print(f'y(1) = {y(1)}')
print(f'a = {a}')
```

```
# y(1) = 12
a = 10
```

Резюме

Подытожим, какие бывают
области видимости в Python:

Локальная область видимости
переменные,
определенные
внутри функций

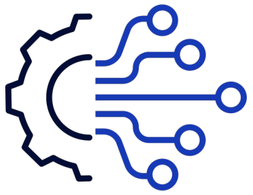
Глобальная область видимости
переменные,
определенные
на глобальном уровне

Встроенная область видимости
переменные, заранее
определенные
в Python

Множественные аргументы и аргументы по умолчанию

Множественные аргументы

Как мы видели ранее, функции могут принимать **1 аргумент** или не принимать аргументы вовсе.



Теперь рассмотрим, как задать функцию с несколькими параметрами:

```
def add_2(a, b):
  return a + b


def add_3(a, b, c):
  return a + b + c
```

y(x) = a * x + b

Продолжим эксперименты с нашей любимой функцией **y(x) = 5 * x + 2**, но теперь, как сказал бы математик, перейдем на новый уровень абстракции – введем параметры **a** и **b**:

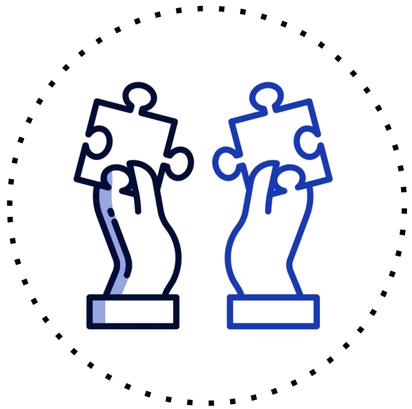
```
def y(x, a, b):
    return a * x + b
```

Что же нам это дает?

y(x) = a * x + b

Теперь мы можем удобно менять **a** и **b**, например, в цикле:

```
def y(x, a, b):
    return a * x + b


for a in range(1, 3):
    for b in range(-1, 1):
        for i in range(0, 3):
            print(f'a = {a}, b = {b} => y = {y(i, a, b)}')
```



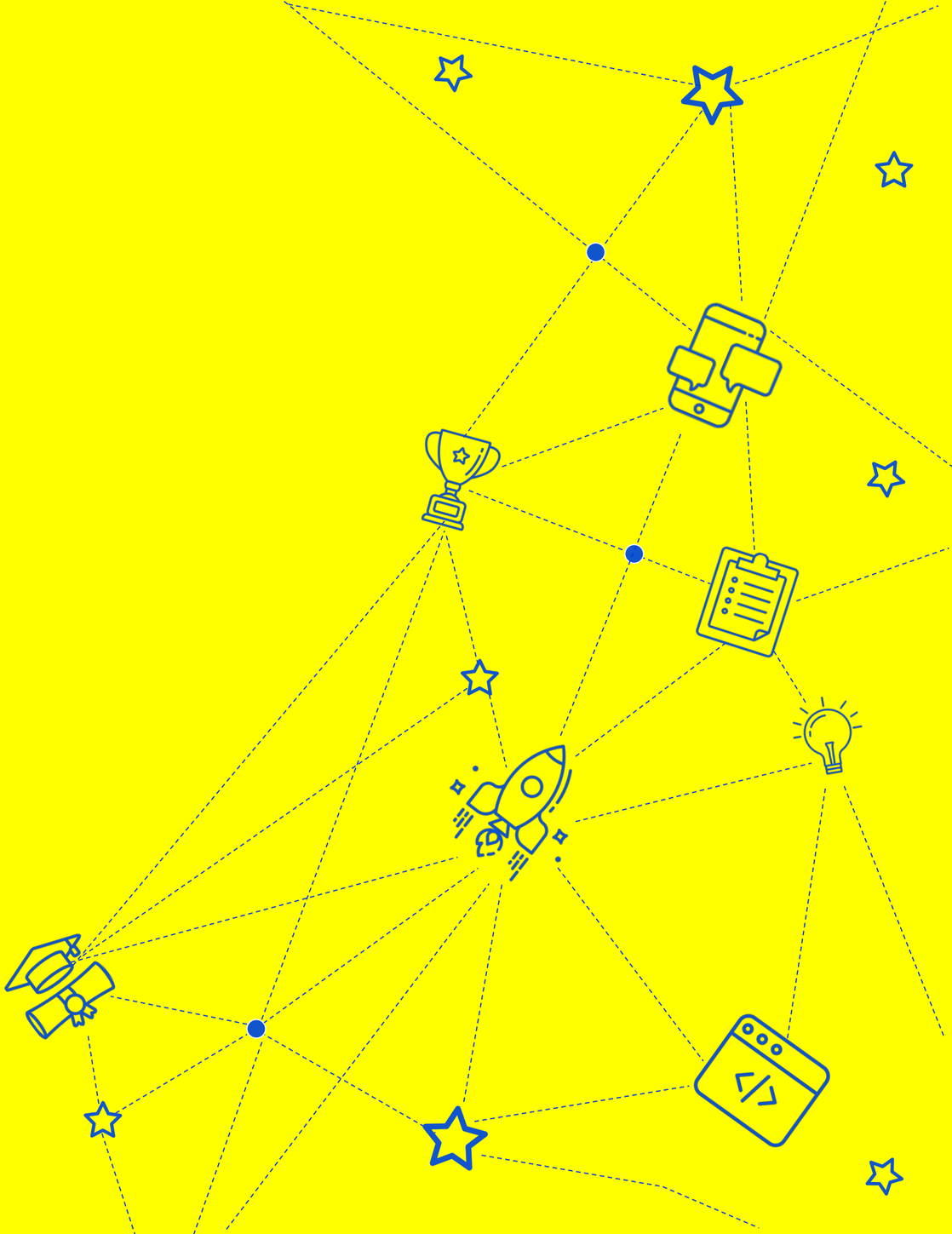
Аргументы по умолчанию

Можно ли сделать еще интереснее?

Конечно!

```
def y(x, a=5, b=2):
    return a * x + b


print(y(1))  # = 7
print(y(1, 10, 4))  # = 14
```

Лямбда-функции



Что такое лямбда-функции

Лямбда-функции – это:

синтаксический сахар,
который используется
«для красоты»

безымянная функция
используется, когда функция короткая,
используется 1 раз – удобно,
не надо придумывать имя

Еще немного математики :)

Рассмотрим нашу до боли знакомую функцию

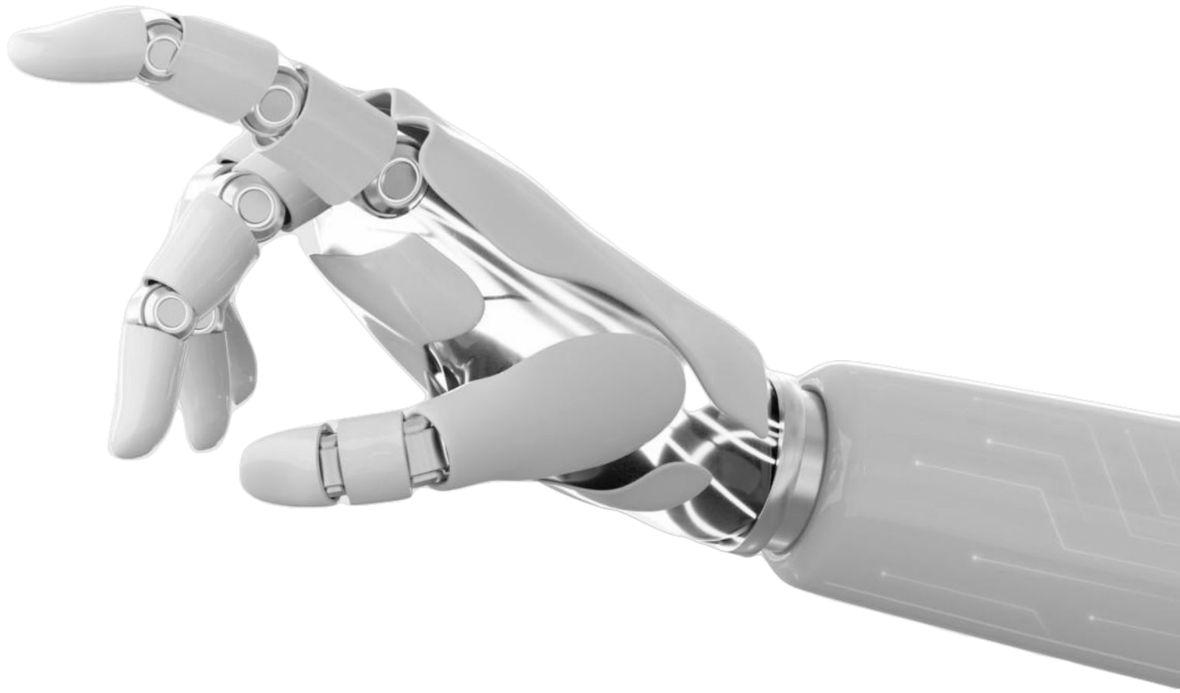
y(x) = 5 * x + 2

Но теперь:

- ★ обозначим **g(x) = 5 * x**
- ★ тогда мы можем переобозначить **y(x) = g(x) + 2**
- ★ а можем даже передавать функцию **g(x)** как параметр, т.е. **y(x, g(x)) = g(x) + 2**

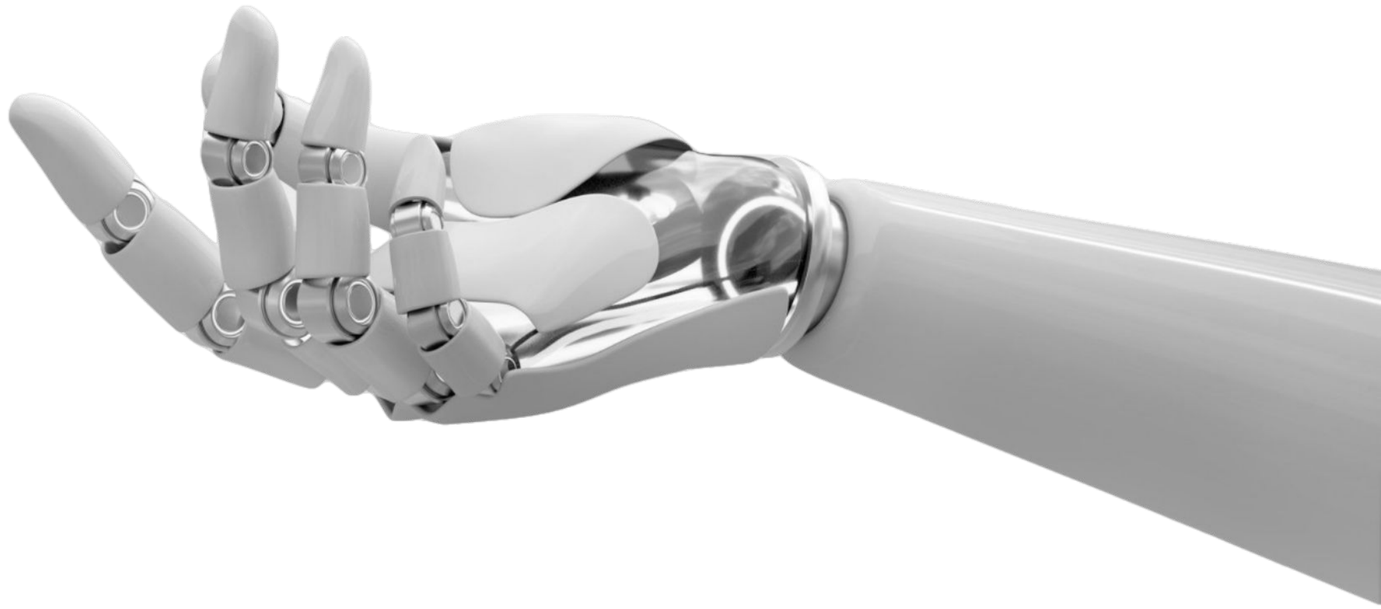
Как это выглядит в Python

```
def g(x):
    return 5 * x


def y(x, g):
    return g(x) + 2


print(f'y = {y(1, g)}')
```



А теперь с лямбдой

```
def y(x, g):
    return g(x) + 2


print(f'y = {y(1, lambda x: 5 * x)}')
```

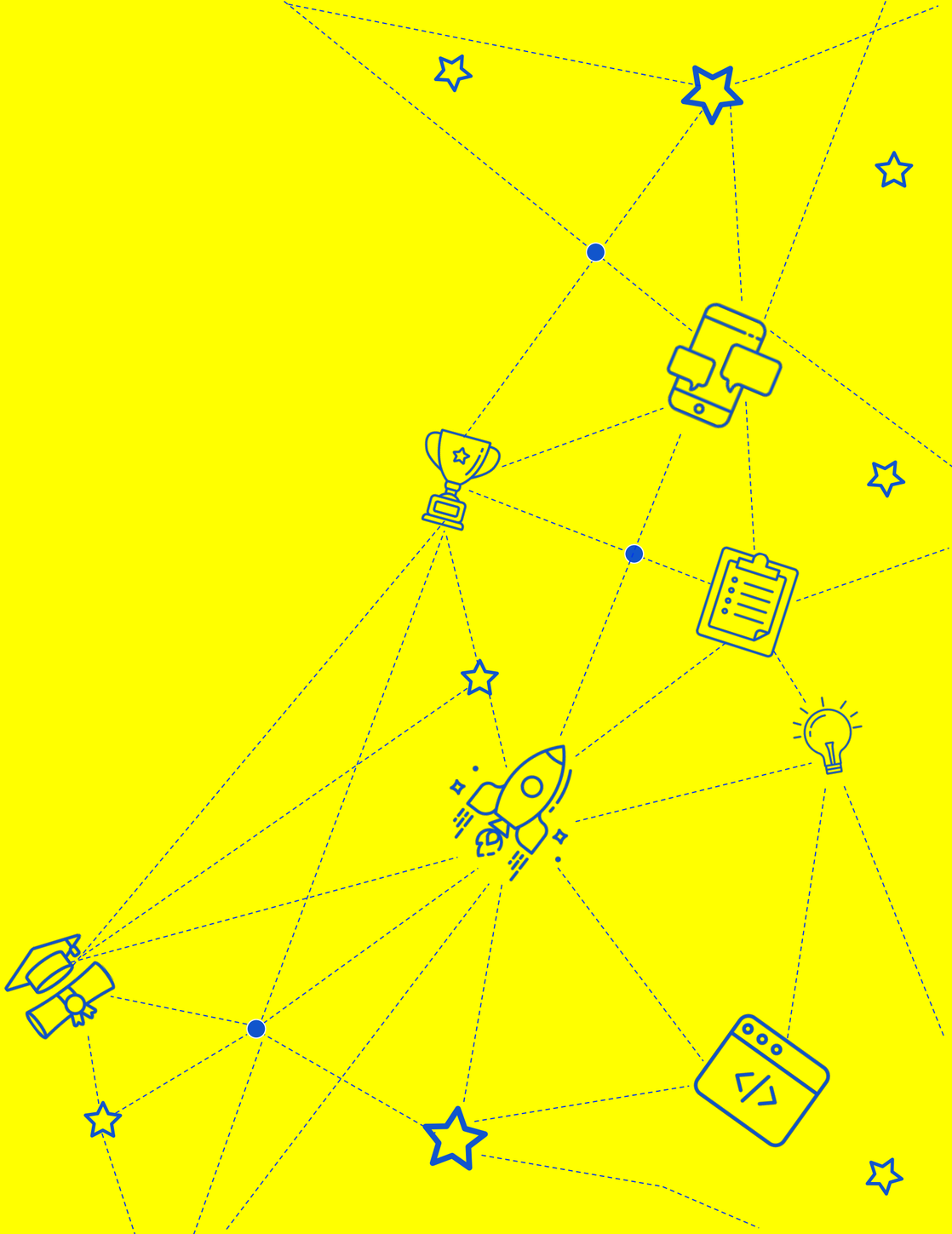


Еще один пример

Здесь мы рассмотрим **лямбду** с двумя аргументами:

```
minimum = lambda a, b: a if a < b else b
print(minimum(3, 5))
```

Заключение



One more thing...

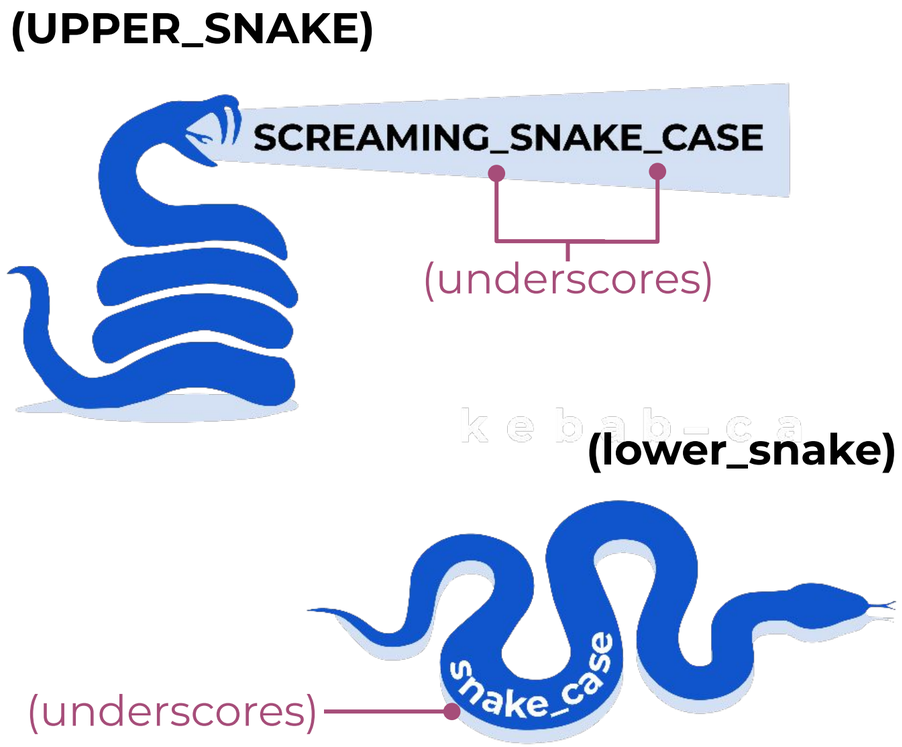
Как выбирать **имена** функций?

snake_case:

- ★ имена должны быть записаны в нижнем регистре;
- ★ слова должны разделяться нижними подчеркиваниями;
- ★ имена не должны начинаться с цифр **0..9**.

Нельзя:

- ★ переопределять встроенные имена;
- ★ давать имена, совпадающие с ключевыми словами языка Python.



СПАСИБО ЗА ВНИМАНИЕ