Анонимные функции lambda

В Python **лямбда-функция** – это анонимная функция, выраженная одним выражением.

- Ее можно использовать вместо некоторой маленькой функции
- Лямбда-функции регулярно используются со встроенными функциями map() и filter(), а также functools.reduce(), представленными в модуле functools

```
>>> easy_sqrt = lambda x: x ** 0.5
>>> easy_sqrt(4)
2.0
>>> list(map(lambda x: x.upper(), ['cat', 'dog', 'cow']))
['CAT', 'DOG', 'COW']
>>> list(filter(lambda x: 'o' in x, ['cat', 'dog', 'cow']))
['dog', 'cow']

```

Встроенные функции

abs()	classmethod()	filter()	id()	max()	property()	super()
all()	compile()	float()	input()	memoryview()	repr()	tuple()
any()	complex()	format()	int()	setattr()	reversed()	type()
ascii()	delattr()	frozenset()	isinstance()	next()	round()	vars()
bin()	dict()	getattr()	issubclass()	object()	set()	zip()
bool()	dir()	globals()	iter()	oct()	slice()	__import__()
bytearray()	divmod()	hasattr()	len()	open()	sorted()	
bytes()	enumerate()	hash()	list()	ord()	staticmethod()	
callable()	eval()	min()	locals()	pow()	str()	
chr()	exec()	hex()	map()	print()	sum()	

Области видимости

В Python действует правило **LEGB**, которым интерпретатор пользуется при поиске переменных:

- **L (Local)** – в локальной (внутри функции)
- **E (Enclosing)** – в локальной области объемлющих функций
- **G (Global)** – в глобальной области видимости
- **B (Built-in)** – во встроенной (Зарезервированные значения Python)

```python
num_1 = 11
num_2 = 15


def sqrt_sum(number_1, number_2):
    pow_n = 2
    print(f"Возводим числа в степень {pow_n}")

    def sqrt(some_number):
        return some_number ** 0.5

    return sqrt(number_1) + sqrt(number_2)


if __name__ == '__main__':
    print(f"Результат: {sqrt_sum(num_1, num_2)}")

```

Пространства имен

Пространства имен – своего рода разделы, внутри которых определенное имя уникально и не связано с такими же именами в других пространствах имен.

В Python есть два пространства имен:

- Локальное
- Глобальное

```python
some_val = 123
print(locals())
print(globals())


def some_func(a=1):
    some_val = 234
    loc_var = 111
    print(locals())
    print(globals())

    def inner_func(c=2):
        some_val = 234
        inner_var = 222
        print(locals())
        print(globals())
        return some_val

    inner_func()

    return some_val

some_func()
```

Локальное и глобальное пространство имен

В Python есть 2 функции, с помощью которых можно посмотреть, что в локальном и глобальном пространствах имен:

- **locals()** – возвращает словарь, который содержит имена и значения локального пространства имен
- **globals()** – возвращает словарь, который содержит имена и значения глобального пространства имен

```
1  global_var = 123
2
3
4  def some_function():
5      local_var = 321
6      print(globals())
7      print(locals())
8
```

Ключевое слово global

Если у нас есть 2 переменные, одна в глобальном пространстве, а вторая в локальном, то

- При присвоении переменной в локальной области – будет создана переменная в локальной области, значение в глобальной области не будет изменено
- Если нам нужно изменять значение из глобальной области, то нужно воспользоваться ключевым словом **global**

Если не использовать global, то Python берет имя из локального пространства имен, работает с локальной переменной и после того, как функция выполнит свою работу, **значение пропадает**

```
1  some_val = 123
2
3
4  def local_val():
5      some_val = 24
6      return some_val
7
8
9  local_val()
10 print(some_val)
11
12
13 def global_val():
14     global some_val
15     some_val = 35
16     return some_val
17
18
19 global_val()
20 print(some_val)
21
```

Рекурсия

Из одной функции можно вызвать другую. Но помимо этого, из функции можно вызвать эту же функцию. Процесс, когда в своем теле функция вызывает себя же, называется рекурсией

```
1  def decode_utf(data):
2      if isinstance(data, bytes) or isinstance(data, bytearray):
3          data = data.decode('utf-8')
4      elif isinstance(data, list):
5          for key, value in enumerate(data):
6              data[key] = decode_utf(value)
7      elif isinstance(data, dict):
8          data = {
9              decode_utf(key): decode_utf(value) for key, value in data.items()
10         }
11     return data
12
```