Функции

что это такое такое, как повторно использовать код

Функция

Функция – именованный фрагмент кода, к которому можно обратиться из другого места программы.

Основная **цель** использование функций – **повторное использование** некоторого кода

С функцией можно сделать 2 вещи:

- Определить
- Вызвать

```python
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1) * n

print(factorial(5))
```

Как определить функцию

Порядок определения функции:

1. Написать ключевое слово «**def**»
2. Написать **имя функции** (имя функции выбираем по тем же правилам, что и имена переменных)
3. **В круглых скобках** указываем **аргументы**, которые должна принимать функция (от 0 до …)
4. Не забываем в конце добавить символ «:»

```
1  def sum_pow(num_1, num_2):
2      return num_1 ** 2 + num_2 ** 2
```

Как вызвать функцию

Вызов функции – выполнение тела с переданными ей аргументами.

Для того, чтобы вызвать функцию из другого участка кода, нужно написать ее имя и круглые скобки (если функции нужны какие-то данные, то пишем их в круглых скобках)

```
1  def count_positive(data: list):
2      counter = 0
3      for item in data:
4          if item > 0:
5              counter += 1
6      return counter
7
8
9  print(count_positive([1, 2, -3, 4, -2]))
10
```

Как вернуть значение из функции

Чтобы вернуть из функции какое-либо значение, используется ключевое слово **return**.

- Функция может возвращать любое количество значений (вернуться внутри tuple)
- Возвращаемое значение можно присвоить переменной для дальнейшего использования
- Если функция ничего не возвращает (мы не написали return), то результат ее выполнения будет None

```
1  def some_func():
2      return 1, 2, 3
3
4
5  def none_func():
6      pass
7
8
9  # 'a': 1, 'b': 2, 'c': 3
10 a, b, c = some_func()
11
12 # 'is_none': None
13 is_none = none_func()
14
```

Чистые функции

Функция с побочным эффектом (side effect) – функция, которая изменяет какие-то внешние данные

Чистая функция – функция, которая получает аргументы, проводит действия, возвращает результат, не затрагивая данные вне себя. Такая функция будет работать каждый раз одинаково при одних и тех же входящих параметрах. Данные функции легче тестировать

```python
from copy import deepcopy
from datetime import datetime

some_data = {
    'username': 'potato',
    'age': 25
}


def function(data: dict):
    data['last_visit'] = datetime.now()


def clear_function(data: dict):
    data_copy = deepcopy(data)
    data_copy['last_visit'] = datetime.now()
    return data_copy


clear_function(some_data)
function(some_data)

```

Внутренние функции

В теле некоторой функции можно определить другую функцию.

- Используется, если необходимо внутри тела выполнять какие то повторяющиеся действия

```
1  def sqrt_all(*args):
2      def sqrt_numb(numb):
3          return numb ** 0.5
4      return [sqrt_numb(item) for item in args]
5
6
7  print(sqrt_all(1, 2, 3, 4, 5))
8
```

Замыкания

Внутренняя функция может действовать как замыкание.

Замыкание – функция, которая динамически генерируется другой функцией и они обе могут изменять и запоминать значения переменных, которые были созданы вне функции.

Замыкания используются в декораторах.

```
1  def multiply(x):
2      def power(y):
3          return x * y
4      return power
5
6
7  x_3 = multiply(3)
8  print(x_3(6))  # 18
9
```

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()	

Аргументы

передача данных в функцию,
разновидности, значения по умолчанию

Аргументы функции

Аргументы функции – значения, которые передаются в функцию.

- Функция может принимать любое количество аргументов (даже неопределенное)
- В Python есть 2 типа передачи аргументов:
  - Позиционный
  - С помощью ключевых слов
- Сперва необходимо определять позиционным ме
  ключевыми словами

```
1  def func(a1, a2, a3, a4, a5=None, a6=None):
2      print(locals())
3
4
5  func(1, 3, a4=2, a3=6, a6=7)
6  # {'a1': 1, 'a2': 3, 'a3': 6,
7  # 'a4': 2, 'a5': None, 'a6': 7}
8
```

Значения по умолчанию

Иногда необходимо, чтобы некоторый аргумент функции был необязательным. Тогда для него указывается значение по умолчанию. Если указано такое значение, то этот аргумент можно не передавать, а вместо него при выполнении будет использовано значение по умолчанию.

- Значение по умолчанию высчитывается, когда функция определяется, а не выполняется. Типичная ошибка – использование изменяемого типа данных

```
1  def say_phrase(text, repeat=1):
2      for i in range(repeat):
3          print(text)
4
5
6  say_phrase('I\'m blue')
7  say_phrase('Da ba dee da ba di', 7)
8
```

Произвольное количество аргументов

Иногда необходимо, чтобы функция могла принимать произвольное количество аргументов. Есть 2 способа это сделать. Первый:

- Указать одним из аргументов «*args». Данный аргумент будет представлен в виде tuple. Если не передать значения, то будет пустой tuple. Не обязательно называть данный аргумент «args», но так принято, так что настоятельно рекомендуется

```
1  def sum_all(*args):
2      print(locals())
3      return sum(args)
4
5
6  result = sum_all(1, 3, 2)
7
```

Произвольное количество аргументов

Второй способ:

- Указать одним из аргументов «**kwargs». Данный аргумент будет представлен в виде dict. Если не передать параметры в виде ключ-значение, то будет пустой dict. Не обязательно называть данный аргумент «kwargs», но так принято, так что настоятельно рекомендуется

```
1  def sum_all(*args):
2      print(locals())
3      return sum(args)
4
5
6  result = sum_all(1, 3, 2)
7
```

Порядок указания аргументов

При объявлении функции:

- Позиционные аргументы
- Аргументы со значением по умолчанию (имя=значение)
- *args
- Аргументы, которые должны передаваться только по ключевым словам (имя=значение)
- **kwargs

```
1  def some_function(arg_1, arg_2=2, *args, arg_3=None, **kwargs):
2      pass
```

Порядок указания аргументов

При вызове функции:

- Любые позиционные аргументы значение
- *итерируемый_объект
- Комбинация любых ключевых аргументов имя=значение
- **словарь

```
1  def some_function(arg_1, arg_2=2, *args, arg_3=None, **kwargs):
2      pass
3
4
5  some_function(1, 2, *(1, 2, 3), arg_3=123, **{'1': 100})
6
```