

Лекция 5. Функции. Исключения в Python

Функции и их назначение. Возврат значений из функции: одно или больше.

Параметры функции. Обязательные и необязательные параметры.

Аргументы функций *args и **kwargs. Порядок параметров.

Исключения try .. except .. finally. Перехват исключений



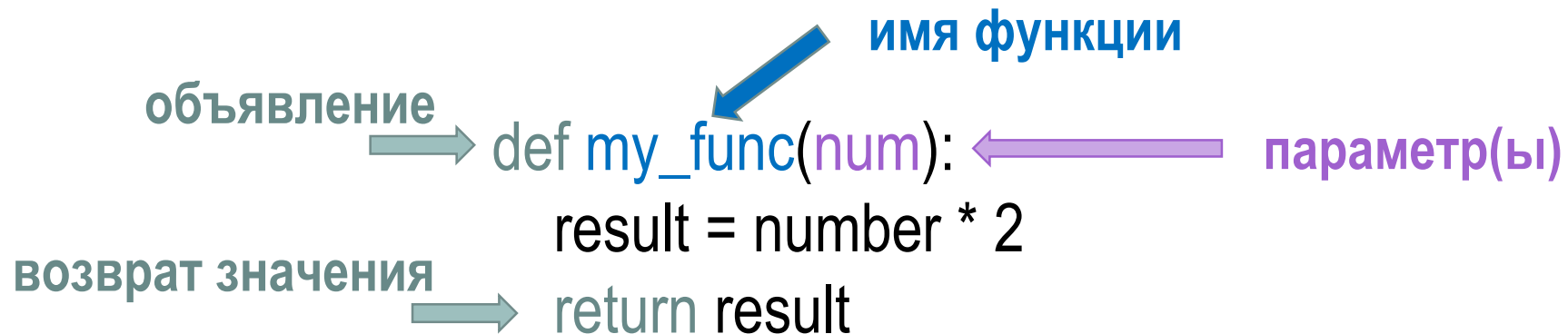


Функции

Функции и их назначение. Возврат значений из функции: одно или больше. Параметры функции. Обязательные и необязательные параметры. Аргументы функций `*args` и `**kwargs`. Порядок параметров.

Функция

Часть кода, которую можно оформить отдельно, дать ей имя и вызывать в нужный момент. Функции обычно возникают, когда какую-то последовательность команд требуется вызывать несколько раз



The diagram illustrates the components of a Python function definition. It shows the code `def my_func(num):` on the first line and `result = number * 2` and `return result` on the second line. Annotations include: a blue arrow pointing to `my_func` labeled "имя функции" (function name); a light blue arrow pointing to the `def` keyword labeled "объявление" (declaration); a purple arrow pointing to `num` labeled "параметр(ы)" (parameter(s)); and a light blue arrow pointing to the `return` statement labeled "возврат значения" (return value).

```
def my_func(num):  
    result = number * 2  
    return result
```



help() и docstring

help() – вызывает справку по нужной функции

```
help(print)
```

docstring (document string) – средство документирования в Python. Сразу после объявления функции можно записать строковое значение. Оно станет документацией

```
def my_func(num):
```

```
    """функция умножения числа на 2"""
```

```
    result = num * 2
```

```
    return result
```



Параметры функции

- ❑ Это одно или больше входных значений, которые передаются в функцию.
- ❑ Функция может быть и без параметров.
- ❑ Для всех параметров можно указать значение по умолчанию. В этом случае параметр можно не писать при вызове

```
▼ def my_func(num, mul=2):  
    print(num, mul)  
    print(num * mul)  
  
my_func(4)  
my_func(4, 3)
```



Переменное количество параметров

Можно объявить функцию с переменным количеством параметров через `*args` или `**kwargs`

- ❑ `*args` – это последовательность аргументов (кортеж)
- ❑ `**kwargs` – это последовательность «название аргумента»: «значение» (словарь)



Переменное количество параметров (продолжение)

***args**

```
▼ def my_func(*args):  
    print(type(args))  
▼ for arg in args:  
    print(arg)  
    print(args[0] * args[1])  
  
my_func(4, 2)
```

****kwargs**

```
▼ def my_func(**kwargs):  
    print(type(kwargs))  
▼ for k,v in kwargs.items():  
    print(k, v)  
    print(kwargs["num"]*kwargs["mul"])  
  
my_func(num=4, mul=2)
```



Возврат значений

- ❑ Значение возвращается через `return`
- ❑ Можно вернуть только одно значение
- ❑ Если нужно вернуть несколько значений – возвращайте коллекцию
- ❑ Если `return` явно не указан, функция все равно возвращает значение - `None`

```
▼ def my_func(num):  
    return num ← 4  
print(my_func(4))
```

```
▼ def my_func(num):  
    pass ← None  
print(my_func(4))
```



Области видимости

- ❑ `local`. По умолчанию. Переменная доступна только внутри функции, где объявлена
- ❑ `global`. Дает доступ к переменной во всей программе на всех уровнях
- ❑ `nonlocal`. Дает доступ к переменной на один уровень выше



Области видимости (продолжение)

global

```
def pay4g():
    global onetime_bonus ← global
    onetime_bonus = 700
    print(f"pay4g - внешние salary({salary}) +
bonus({bonus}) + global onetime_bonus({onetime_bonus}):
{salary + bonus + onetime_bonus}")

print(f"До вызова pay4g onetime_bonus={onetime_bonus}")
pay4g()
print(f"После вызова pay4g onetime_bonus=
{onetime_bonus}")
```

```
До вызова pay4g onetime_bonus=300
pay4g - внешние salary(500) + bonus(500) +
global onetime_bonus(700): 1700
После вызова pay4g onetime_bonus=700
```

nonlocal

```
onetime_bonus = 300 ← ВНЕШНЯЯ
is_heroic = True
def pay5nl():
    onetime_bonus = 0 ← local
    def heroic():
        if is_heroic:
            nonlocal onetime_bonus ← nonlocal
            onetime_bonus = 700
    heroic()
    print(f"pay5nl - внешние salary({salary}) + bonus({bonus}) + nonlocal
onetime_bonus({onetime_bonus}): {salary + bonus + onetime_bonus}")

print(f"До вызова pay5nl onetime_bonus={onetime_bonus}")
pay5nl()
print(f"После вызова pay5nl onetime_bonus={onetime_bonus}")
```

```
До вызова pay5nl onetime_bonus=300
pay5nl - внешние salary(500) + bonus(500) +
nonlocal onetime_bonus(700): 1700
После вызова pay5nl onetime_bonus=300
```





Исключения

Исключения `try .. except .. finally`. Перехват исключений

Исключения

Исключение — возникновение неустранимой ошибки во время выполнения кода, приводящее к падению программы. Исключения можно перехватывать в самом программном коде

```
try ... except ... finally
```



Исключения (продолжение)

try:

«опасный» код

except:

код, который выполняется при исключении

finally:

завершающий код, который выполняется всегда



Исключения (продолжение)

без try ... except

```
budget = 1000
def pay_ex():
    dangerous_coeff = 0
    return budget/dangerous_coeff
print(pay_ex())
```

```
Traceback (most recent call last):
  File "main.py", line 104, in <module>
    print(pay_ex())
  File "main.py", line 103, in pay_ex
    return budget/dangerous_coeff
ZeroDivisionError: division by zero
```

с try ... except

```
budget = 1000
def pay_ex():
    dangerous_coeff = 0
    return budget/dangerous_coeff

try:
    pay_ex()
    print("Вызвали pay_ex")
except Exception as e:
    print(e)
finally:
    print("Всегда будет работать")
```

```
division by zero
Всегда будет работать
```



Иерархия исключений в Python

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
    |   +-- FloatingPointError
    |   +-- OverflowError
    |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
```

```
+-- ImportError
    |   +-- ModuleNotFoundError
+-- LookupError
    |   +-- IndexError
    |   +-- KeyError
+-- MemoryError
+-- NameError
    |   +-- UnboundLocalError
+-- OSError
    |   +-- BlockingIOError
    |   +-- ChildProcessError
```



Создание исключений

- **raise**. Вызывает конкретное исключение
- **assert**. Вызывает исключение, если условие не выполняется



Создание исключений (raise)

```
raise ZeroDivisionError
```

```
try:  
    raise ZeroDivisionError  
except:  
    print("Поймали исключение")
```

```
Traceback (most recent call last):  
  File "main.py", line 114, in <module>  
    raise ZeroDivisionError  
ZeroDivisionError
```

Поймали исключение



Создание исключений (assert)

```
assert 5 in [1, 2, 3, 4]
```

```
Traceback (most recent call last):  
  File "main.py", line 121, in <module>  
    assert 5 in [1, 2, 3, 4]  
AssertionError
```

```
try:  
    assert 5 in [1, 2, 3, 4]  
except:  
    print("Поймали исключение")
```

Поймали исключение

