Основы языка программирования Python

Функции

```python
def hello_world():
    """Функция без параметров. Возвращает None."""
    print("Hello, world!")

def hello(name):
    """Функция с одним параметром. Возвращает None."""
    print(f"Hello, {name}")

def add(a, b):
    """Функция с двумя параметрами."""
    c = a + b
    return c

def add_sub(a, b):
    """Функция с двумя параметрами.
    Возвращает кортеж из двух значений."""
    s = a + b
    d = a - b
    return (s, d)
```

```python
# Демонстрация "утиной" типизации
def add(a, b):
    return a + b

foo = add(10, 20)
bar = add(3+2j, 20.5)
baz = add("hello ", "world")
spam = add([10, 20, 30], ["hello", "world"])

print(f"{foo=}", f"{bar=}", f"{baz=}", f"{spam=}", sep="\n")
```

```
foo=30
bar=(23.5+2j)
baz='hello world'
spam=[10, 20, 30, 'hello', 'world']
```

```python
# Демонстрация "утиной" типизации
def add(a, b):
    return a + b

# Ошибка!
foo = add(10, "hello")
```

```
Traceback (most recent call last):
  File ".../example_error_01.py", line 6, in <module>
    foo = add(10, "hello")
          ^^^^^^^^^^^^^^^^
  File ".../example_error_01.py", line 3, in add
    return a + b
           ~~^~~
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```python
# Демонстрация ошибок работы с функциями
def add(a, b):
    return a + b

# Ошибка! Не указан обязательный параметр
foo = add(10)
```

```
Traceback (most recent call last):
  File "../example_error.py", line 6, in <module>
    foo = add(10)
          ^^^^^^^
TypeError: add() missing 1 required positional argument: 'b'
```

```python
# Демонстрация ошибок работы с функциями
def add(a, b):
    return a + b

# Ошибка! Передается лишний параметр
foo = add(10, 20, 30)
```

```
Traceback (most recent call last):
  File ".../example_error.py", line 6, in <module>
    foo = add(10, 20, 30)
          ^^^^^^^^^^^^^^^
TypeError: add() takes 2 positional arguments but 3 were given
```

```python
# Все эти функции возвращают None

def hello_world_1():
    print("Hello, world - 1!")

def hello_world_2():
    print("Hello, world - 2!")
    return None

def hello_world_3():
    print("Hello, world - 3!")
    return

foo = hello_world_1()
bar = hello_world_2()
baz = hello_world_3()

print(f"{foo=}")
print(f"{bar=}")
print(f"{baz=}")
```

```
Hello, world - 1!
Hello, world - 2!
Hello, world - 3!
foo=None
bar=None
baz=None
```

```python
from math import sqrt

def sum_sqrt(x, y):
    """Пример функции, которая возвращает
    разные типы в зависимости от условий."""
    if x + y < 0:
        return

    return sqrt(x + y)

foo = sum_sqrt(2, 7)
bar = sum_sqrt(2, -7)

print(f"{foo=}")
print(f"{bar=}")
```

```
foo=3.0
bar=None
```

```python
# Пример с распаковкой возвращаемых значений

def add_sub(a, b):
    """Функция с двумя параметрами.
    Возвращает кортеж из двух значений."""
    s = a + b
    d = a - b
    return (s, d)

spam = add_sub(10, 3)
foo, bar = add_sub(10, 3)

print(f"{spam=}")
print(f"{foo=}")
print(f"{bar=}")
```

```
spam=(13, 7)
foo=13
bar=7
```

```python
# Использование именованных параметров

def sub(a, b):
    return a - b

foo = sub(30, 10)
bar = sub(a=30, b=10)
baz = sub(b=10, a=30)
spam = sub(30, b=10)

print(f"{foo=}")
print(f"{bar=}")
print(f"{baz=}")
print(f"{spam=}")
```

```
foo=20
bar=20
baz=20
spam=20
```

```python
# Использование именованных параметров

def sub(a, b):
    return a - b

# Ошибка!
# Позиционные параметры должны быть указаны до именованных
eggs = sub(a=30, 10)
```

```
  File ".../example_error.py", line 8
    eggs = sub(a=30, 10)
                     ^
SyntaxError: positional argument follows keyword argument
```

```python
# Функции с параметрами по умолчанию
def mul(a, b=2):
    return a * b

foo = mul(3, 4)
bar = mul(3)
spam = mul(3, b=4)

print(f"{foo=}")
print(f"{bar=}")
print(f"{spam=}")
```

```
foo=12
bar=6
spam=12
```

```
# Ошибка!
# Параметры со значениями по умолчанию
# должны располагаться в конце списка параметров
def mul(a = 1, b):
    return a * b
```

```
 File ".../example_error.py", line 4
    def mul(a = 1, b):
                   ^
SyntaxError: non-default argument follows default argument
```

```python
# Функции с параметрами по умолчанию
def mul(a=1, b=2):
    return a * b

foo = mul()
bar = mul(2, 3)
spam = mul(b=3)

print(f"{foo=}")
print(f"{bar=}")
print(f"{spam=}")
```

```
foo=2
bar=6
spam=3
```

```python
# Функция с переменным числом позиционных параметров
def mul(a, *args):
    print(f"mul: {args=}")
    result = a
    for x in args:
        result *= x

    return result

foo = mul(2, 3, 4, 5)
bar = mul(3)

print(f"{foo=}")
print(f"{bar=}")
```

```
mul: args=(3, 4, 5)
mul: args=()
foo=120
bar=3
```

```python
# Функция с переменным числом именованных параметров
def calculate(x, **kwargs):
    print(f"{kwargs=}")
    result = x
    if 'mul' in kwargs:
        result *= kwargs['mul']

    if 'add' in kwargs:
        result += kwargs['add']

    return result

print(f'{calculate(5)=}', end="\n\n")
print(f'{calculate(10, mul=2)=}', end="\n\n")
print(f'{calculate(10, add=5)=}', end="\n\n")
print(f'{calculate(10, add=5, mul=2)=}', end="\n\n")
print(f'{calculate(x=10, mul=2, add=5)=}', end="\n\n")
```

```
kwargs={}
calculate(5)=5

kwargs={'mul': 2}
calculate(10, mul=2)=20

kwargs={'add': 5}
calculate(10, add=5)=15

kwargs={'add': 5, 'mul': 2}
calculate(10, add=5, mul=2)=25

kwargs={'mul': 2, 'add': 5}
calculate(x=10, mul=2, add=5)=25
```

```python
# Функция с переменным числом именованных параметров
# Более компактная версия
def calculate(x, **kwargs):
    print(f"{kwargs=}")
    result = x
    result *= kwargs.get('mul', 1)
    result += kwargs.get('add', 0)
    return result

print(f'{calculate(5)=}', end="\n\n")
print(f'{calculate(10, mul=2)=}', end="\n\n")
print(f'{calculate(10, add=5)=}', end="\n\n")
print(f'{calculate(10, add=5, mul=2)=}', end="\n\n")
print(f'{calculate(x=10, mul=2, add=5)=}', end="\n\n")
```

```
kwargs={}
calculate(5)=5

kwargs={'mul': 2}
calculate(10, mul=2)=20

kwargs={'add': 5}
calculate(10, add=5)=15

kwargs={'add': 5, 'mul': 2}
calculate(10, add=5, mul=2)=25

kwargs={'mul': 2, 'add': 5}
calculate(x=10, mul=2, add=5)=25
```

Разделительные параметры функции: / и *

Пример из библиотеки Numpy

```
arctan2(x1, x2, /, out=None, *, where=True,
        casting='same_kind', order='K', dtype=None, subok=True)
```

```
def func(pos1, ...,posN, /,any1, ..., anyN, *, kwd1, ..., kwdN):
```

Назначение разделительных параметров

Левая сторона	Разделитель	Правая сторона
Только позиционные параметры	/	Позиционные или именованные параметры
Позиционные или именованные параметры	*	Только именованные параметры

```
arctan2(x1, x2, /, out=None, *, where=True,
        casting='same_kind', order='K', dtype=None, subok=True)
```

Примеры правильных вызовов:

```
foo = arctan2(0.3, 0.5)
foo = arctan2(0.3, 0.5, None)
foo = arctan2(0.3, 0.5, out=None)
foo = arctan2(0.3, 0.5, None, where=True)
foo = arctan2(0.3, 0.5, out=None, where=True)
```

```
arctan2(x1, x2, /, out=None, *, where=True,
        casting='same_kind', order='K', dtype=None, subok=True)
```

Примеры вызовов с ошибками:

```
foo = arctan2(0.3, x2=0.5)
foo = arctan2(x1=0.3, x2=0.5)
foo = arctan2(0.3, 0.5, None, True)
foo = arctan2(0.3, 0.5, out=None, True)
```

```python
# Функция с разделительным параметром /

def func(a, b, /, action):
    if action == "add":
        return a + b
    elif action == "mul":
        return a * b

foo = func(2, 3, "add")
bar = func(2, 3, action="add")

print(f"{foo=}")
print(f"{bar=}")
```

```
foo=5
bar=5
```

```python
# Функция с разделительным параметром /

def func(a, b, /, action):
    if action == "add":
        return a + b
    elif action == "mul":
        return a * b

# Ошибка!
# Указание именованных параметров до разделительного параметра /
foo = func(a=2, b=3, action="add")
```

```
Traceback (most recent call last):
  File ".../example_error.py", line 11, in <module>
    foo = func(a=2, b=3, action="add")
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: func() got some positional-only arguments passed as keyword arguments: 'a, b'
```

```python
# Функция с разделительным параметром *

def func(a, b, *, action):
    if action == "add":
        return a + b
    elif action == "mul":
        return a * b

foo = func(2, 3, action="add")
bar = func(a=2, b=3, action="add")

print(f"{foo=}")
print(f"{bar=}")
```

```
foo=5
bar=5
```

```python
# Функция с разделительным параметром *

def func(a, b, *, action):
    if action == "add":
        return a + b
    elif action == "mul":
        return a * b

# Ошибка!
# Указание позиционных параметров после разделительного параметра *
foo = func(2, 3, "add")
```

```
Traceback (most recent call last):
  File ".../example_error.py", line 11, in <module>
    foo = func(2, 3, "add")
          ^^^^^^^^^^^^^^^^^
TypeError: func() takes 2 positional arguments but 3 were given
```

```python
def func(a, b, /, action, *, use_abs=False):
    if action == "add":
        result = a + b
    elif action == "mul":
        result = a * b
    else:
        return None

    if use_abs:
        result = abs(result)

    return result

foo = func(2, -5, "add")
bar = func(2, -5, action="add")
baz = func(2, -5, action="add", use_abs=True)

print(f"{foo=}")
print(f"{bar=}")
print(f"{baz=}")
```

```
foo=-3
bar=-3
baz=3
```

```python
def func(a, b, /, action, *, use_abs=False):
    if action == "add":
        result = a + b
    elif action == "mul":
        result = a * b
    else:
        return None

    if use_abs:
        result = abs(result)

    return result

# Ошибка!
# Указание позиционных параметров после разделительного параметра *
foo = func(2, -5, "add", True)
```

```
Traceback (most recent call last):
  File ".../example_error.py", line 16, in <module>
    foo = func(2, -5, "add", True)
          ^^^^^^^^^^^^^^^^^^^^^^^
TypeError: func() takes 3 positional arguments but 4 were given
```

Функции как объекты

Функции — это объекты

```
>>> def add(a, b): return a + b
>>> type(add)
<class 'function'>
>>> other_calc = add
>>> other_calc(4, 2)
6
>>> add is other_calc
True
```

```
>>> dir(add)
```

['__annotations__', '__builtins__', '__call__', '__class__', '__closure__', '__code__', '__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__getstate__', '__globals__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']

```
>>> add.__call__(4, 2)
6
```

```python
def calc(action, a, b):
    result = action(a, b)
    action_name = action.__name__
    print(f"{action_name}({a}, {b}) = {result}")
    return result

def add(a, b): return a + b

def mul(a, b): return a * b

foo = calc(add, 2, 3)
print(f"{foo=}")
bar = calc(mul, 2, 3)
print(f"{bar=}")
```

```
add(2, 3) = 5
foo=5
mul(2, 3) = 6
bar=6
```

Анонимные функции (лямбда-выражения)


```
lambda arg1, arg2, …, argN: выражение
```

```python
def calc(action, a, b):
    result = action(a, b)
    action_name = action.__name__
    print(f"{action_name}({a}, {b}) = {result}")
    return result

foo = calc(lambda x, y: x + y, 2, 3)
bar = calc(lambda x, y: x * y, 2, 3)
baz = calc(lambda x, y: abs(x + y), 2, -5)
spam = calc(lambda x, y: abs(x * y), 2, -5)
```

```
<lambda>(2, 3) = 5
<lambda>(2, 3) = 6
<lambda>(2, -5) = 3
<lambda>(2, -5) = 10
```

```python
def calc(action, a, b):
    result = action(a, b)
    action_name = action.__name__
    print(f"{action_name}({a}, {b}) = {result}")
    return result

add = lambda x, y: x + y
mul = lambda x, y: x * y
add_abs = lambda x, y: abs(x + y)
mul_abs = lambda x, y: abs(x * y)

foo = calc(add, 2, 3)
bar = calc(mul, 2, 3)
baz = calc(add_abs, 2, -5)
spam = calc(mul_abs, 2, -5)
```

```
<lambda>(2, 3) = 5
<lambda>(2, 3) = 6
<lambda>(2, -5) = 3
<lambda>(2, -5) = 10
```

Сортировка списков с помощью метода list.sort()


```
list.sort(*, key=None, reverse=False)
```

```python
# Сортировка слов в тексте

text = "LOREM Ipsum dolor SIT amet Consectetur adipiscing Elit"

words = text.split(" ")
words.sort()

print(f"{words=}")
```

```
words=['Consectetur', 'Elit', 'Ipsum',
'LOREM', 'SIT', 'adipiscing', 'amet', 'dolor']
```

```python
# Сортировка слов в тексте

def tolower(text):
    return text.lower()

text = "LOREM Ipsum dolor SIT amet Consectetur adipiscing Elit"

words = text.split(" ")
words.sort(key=tolower)

print(f"{words=}")
```

```
words=['adipiscing', 'amet', 'Consectetur',
'dolor', 'Elit', 'Ipsum', 'LOREM', 'SIT']
```

```python
# Сортировка элементов списка по второму элементу кортежа

def get_second(items):
    return items[1]

items = [(10, 10), (0, -2), (2, -3), (15, 12), (15, 0)]
items.sort(key=get_second)

print(f"{items=}")
```

```
items=[(2, -3), (0, -2), (15, 0), (10, 10),
(15, 12)]
```

```python
# Использование анонимной функции (lambda)

items = [(10, 10), (0, -2), (2, -3), (15, 12), (15, 0)]
items.sort(key=lambda items: items[1])

print(f"{items=}")
```

```
items=[(2, -3), (0, -2), (15, 0), (10, 10),
(15, 12)]
```

```python
# Использование docstring

def add(a, b):
    """Функция возвращает сумму двух переменных"""
    return a + b

print("add.__doc__:", add.__doc__)
```

```
add.__doc__: Функция возвращает сумму двух переменных
```

Typing.
Указание ожидаемых типов переменных

```python
def mul(a, b):
    return a * b

foo = mul(3.5, 2.0)
bar = mul("spam", 3)

print(f"{foo=}")
print(f"{bar=}")
```

```
foo=7.0
bar='spamspamspam'
```

```python
def mul(a: float, b: float) -> float:
    return a * b

foo = mul(3.5, 2.0)
bar = mul("spam", 3)

print(f"{foo=}")
print(f"{bar=}")
```

```
foo=7.0
bar='spamspamspam'
```

Mypy

Mypy: Static Typing for Python

https://mypy-lang.org

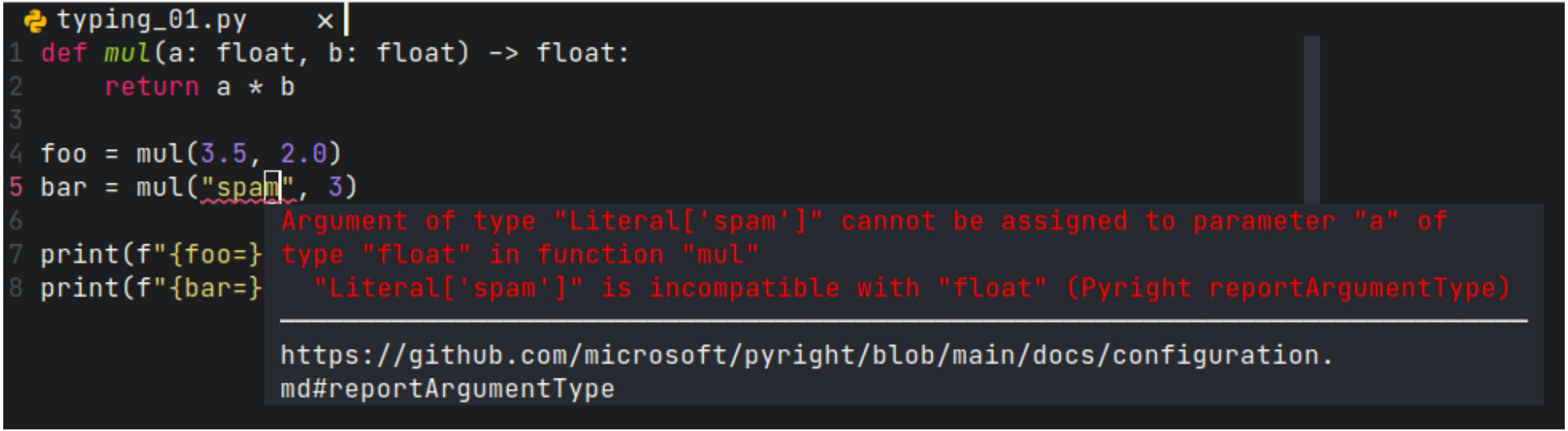
https://github.com/python/mypy

Mypy is an optional static type checker for Python that aims to combine the benefits of dynamic (or "duck") typing and static typing. Mypy combines the expressive power and convenience of Python with a powerful type system and compile-time type checking. Mypy type checks standard Python programs; run them using any Python VM with basically no runtime overhead.

Пример использования Муру

> mypy example_typing_01.py

example_typing_01.py:5: error: Argument 1 to "mul" has incompatible type "str"; expected "float" [arg-type]
Found 1 error in 1 file (checked 1 source file)


typing_01.py
def mul(a: float, b: float) -> float:
return a * b
foo = mul(3.5, 2.0)
bar = mul("spam", 3)
print(f"{foo=}
print(f"{bar=}
Argument of type "Literal['spam']" cannot be assigned to parameter "a" of type "float" in function "mul"
"Literal['spam']" is incompatible with "float" (Pyright reportArgumentType)
https://github.com/microsoft/pyright/blob/main/docs/configuration.md#reportArgumentType

```python
def mul(a: str, b: int) -> str:
    return a * b

bar = mul("spam", 3)

print(f"{bar=}")
```

python language/src/10. Functions/example_typing_02.py

```python
# В качестве первого параметра у функции
# ожидается строка или целое число.
# Функция может вернуть строку или целое число
def mul(a: str | int, b: int) -> str | int:
    return a * b

foo = mul(10, 3)
bar = mul("spam", 3)
```

python language/src/10. Functions/example_typing_03.py

```python
from typing import Union

# В качестве первого параметра у функции
# ожидается строка или целое число.
# Функция может вернуть строку или целое число
def mul(a: Union[str, int], b: int) -> Union[str, int]:
    return a * b

foo = mul(10, 3)
bar = mul("spam", 3)
```

python language/src/10. Functions/example_typing_04.py

```python
import math

def spam(a: float, b: float) -> None | float:
    if a + b >= 0:
        return math.sqrt(a + b)
    return None


foo = spam(4, 5)
bar = spam(-10, 1)

print(f"{foo=}")
print(f"{bar=}")
```

```
foo=3.0
bar=None
```

python language/src/10. Functions/example_typing_05.py

```python
import math
from typing import Optional

def spam(a: float, b: float) -> Optional[float]:
    if a + b >= 0:
        return math.sqrt(a + b)
    return None


foo = spam(4, 5)
bar = spam(-10, 1)
```

python language/src/10. Functions/example_typing_06.py

```python
def spam(foo: list[int]) -> dict[int, bool]:
    return {n: n % 2 == 0 for n in foo}

foo = spam([10, 11, 12, 14, 17])
print(f"{foo=}")
```

```
foo={10: True, 11: False, 12: True, 14: True, 17: False}
```

```python
from typing import Dict, List

def spam(foo: List[int]) -> Dict[int, bool]:
    return {n: n % 2 == 0 for n in foo}

foo = spam([10, 11, 12, 14, 17])
print(f"{foo=}")
```

```
foo={10: True, 11: False, 12: True, 14: True, 17: False}
```

```python
def spam(foo: list[int]) -> dict[int, bool]:
    return {n: n % 2 == 0 for n in foo}

# Результат вызова mypy:
# example_typing_09.py:4: error: Argument 1 to "spam"
# has incompatible type "set[int]"; expected "list[int]"  [arg-type]
# Found 1 error in 1 file (checked 1 source file)
foo = spam({10, 11, 12, 14, 17})
print(f"{foo=}")
```

```
foo={17: False, 10: True, 11: False, 12: True, 14: True}
```

```python
from collections.abc import Iterable

def spam(foo: Iterable[int]) -> dict[int, bool]:
    return {n: n % 2 == 0 for n in foo}

foo = spam({10, 11, 12, 14, 17})
print(f"{foo=}")
```

```
foo={17: False, 10: True, 11: False, 12: True, 14: True}
```