Функции

Докладчик: Евграфов Михаил

Функции

Пользовательские функции

идентификатор

параметры

```
def function_identifier(param1, param2):
    statement1
    statement2
    ...
```

тело функции

вызов функции

```
function_identifier(arg1,
arg2)
```

Символические таблицы

```
def test(arg1: int) -> None:
    local_var = 5
    print(locals())


>>> test(42)
{'arg1': 42, 'local_var': 5}
```

Поиск имен

```
var_global = 42

def enclose_function() -> None:
    var_nonlocal = 142
    def nested_function() -> None:
        var_local = 242
        print(var_local)
        print(var_nonlocal)
        print(var_global)

    nested_function()

>>> enclose_function()
242
142
42
```

Возвращаемое значение

```
def print_object_id(obj: object) -> None:
    obj_id = id(obj)
    print(f"{obj_id = }")

print(print_object_id(5))
# obj_id = 140706553521064
# None
```

return

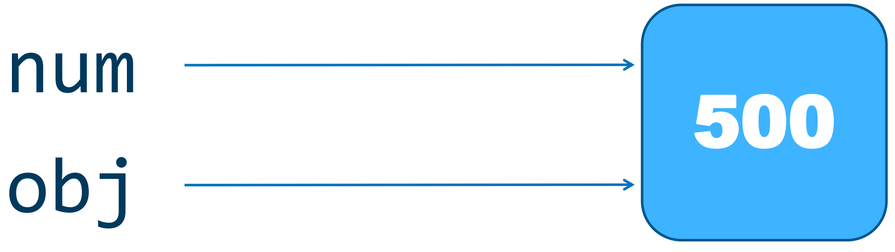
```
def print_number(num: int) -> int:
    print(f"{num = }")
    return num
    print("This message will never be printed")

num = print_number(5)
# num = 5
```

Передача аргументов в функцию

```
def print_object_id(obj: object) -> None:
    obj_id = id(obj)
    print(f"{obj_id = }")

num = 500
print_object_id(num)
```



Передача изменяемых объектов

```
def modify_list(lst: list) -> None:
    print(f"local lst id: {id(lst)}")
    lst.append(42)

>>> lst = [1, 2, 3]
>>> print(f"global lst id: {id(lst)}")
global lst id: 1574609233664
>>> modify_list(lst)
local lst id: 1574609233664
>>> lst
[1, 2, 3, 42]
```

Параметры по умолчанию

```
def function_identifier(
    param1: int, param2: int = 5
) -> int:
    ...

# param1 == 1, param2 == 2
function_identifier(1, 2)

# param1 == 3, param2 == 5
function_identifier(3)
```

Время определения

```
def append_into_list(num: int, lst: list[int] = []) -> None:
    lst.append(num)
    return lst

>>> print(append_into_list(1))
[1]
>>> print(append_into_list(2))
[1, 2]
>>> print(append_into_list(3))
[1, 2, 3]
```

Выход #1

```
def append_into_list(
    num: int, lst: Optional[list[int]] = None
) -> None:
    if lst is None:
        lst = []
    lst.append(num)
    return lst

>>> print(append_into_list(1))
[1]
>>> print(append_into_list(2))
[2]
>>> print(append_into_list(3))
[3]
```

Выход #2

```
_sentinel = object()

def append_into_list(num: int, lst: list[int] = _sentinel) -> None:
    if lst is _sentinel:
        lst = []
    lst.append(num)
    return lst

>>> print(append_into_list(1))
[1]
>>> print(append_into_list(2))
[2]
>>> print(append_into_list(3))
[3]
```

Именованный формат передачи

```
def print_args(
    arg1: int, arg2: int = 2, arg3: int = 3
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        f"{arg3 = }",
        sep="\n",
    )

>>> print_args(arg1=1)
arg1 = 1
arg2 = 2
arg3 = 3
```

Именованный формат передачи

```
def print_args(
    arg1: int, arg2: int = 2, arg3: int = 3
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        f"{arg3 = }",
        sep="\n",
    )

>>> print_args(arg3=5, arg1=6)
arg1 = 6
arg2 = 2
arg3 = 5
```

Именованный формат передачи

```
def print_args(
    arg1: int, arg2: int = 2, arg3: int = 3
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        f"{arg3 = }",
        sep="\n",
    )

>>> print_args(1, arg2=5)
arg1 = 1
arg2 = 5
arg3 = 3
```

Именованный формат передачи

```
def print_args(
    arg1: int, arg2: int = 2, arg3: int = 3
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        f"{arg3 = }",
        sep="\n",
    )

>>> print_args(arg2=3, 1)
...
SyntaxError: positional argument follows keyword argument
```

Именованный формат передачи

```
def print_args(
    arg1: int, arg2: int = 2, arg3: int = 3
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        f"{arg3 = }",
        sep="\n",
    )

>>> print_args(1, arg1=5)
...
TypeError: print_args() got multiple values for argument 'arg1'
```

Строго позиционные параметры

```
def position_only_args(
    arg1: int, arg2: int, /
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        sep="\n",
    )

>>> position_only_args(1, 2)
arg1 = 1
arg2 = 2
```

Строго позиционные параметры

```
def position_only_args(
    arg1: int, arg2: int, /
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        sep="\n",
    )

>>> position_only_args(arg1=1, arg2=2)
...
TypeError: position_only_args() got some positional-only
arguments passed as keyword arguments: 'arg1, arg2'
```

Строго именованные параметры

```
def keward_only_args(
    *, arg1: str, arg2: str
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        sep="\n",
    )

>>> keward_only_args(arg1="arg1", arg2="arg2")
arg1 = 'arg1'
arg2 = 'arg2'
```

Строго именованные параметры

```
def keward_only_args(
    *, arg1: str, arg2: str
) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        sep="\n",
    )

>>> keward_only_args("arg1", arg2="arg2")
...
TypeError: keward_only_args() takes 0 positional
arguments but 1 positional argument were given
```

Комбинированный подход

```
def mixed_args(
    arg1: int, /, arg2: str, *, arg3: str
) -> None:
    ...

>>> mixed_args(1, "arg2", "arg3")                    # ERROR
>>> mixed_args(1, "arg2", arg3="arg3")               # OK
>>> mixed_args(1, arg2="arg2", arg3="arg3")          # OK
>>> mixed_args(arg1=1, arg2="arg2", arg3="arg3")     # ERROR
```

args

```
def concatinate(*args, sep: str = ", ") -> str:
    print(args)
    return sep.join(args)

>>> concatinate("a", "b", "c")
('a', 'b', 'c')
'a, b, c'
```

kwargs

```
def concatinate(sep: str =", ", **kwargs) -> str:
    print(kwargs)
    return sep.join(kwargs.values())

>>> concatinate(a="a", b="b", c="c")
{'a': 'a', 'b': 'b', 'c': 'c'}
'a, b, c'
```

Распаковка

```
def print_args(arg1: int, arg2: int) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        sep="\n",
    )

>>> print_args(*[1, 2])
arg1 = 1
arg2 = 2
```

Распаковка

```
def print_args(arg1: int, arg2: int) -> None:
    print(
        f"{arg1 = }",
        f"{arg2 = }",
        sep="\n",
    )

>>> print_args(**{"arg1": 1, "arg2": 2})
arg1 = 1
arg2 = 2
```

Аннотации типов

пример игнорирования аннотаций типов

```
def print_number(num: int) -> None:
    print(f"{num = }")

print_number(num="123")
```

docstring

```
def do_staff(param1: int, param2: str = "") -> None:
    """
    Do some really usefull staff.

    Args:
        param1: first parameter - integer.
        param2: second parameter - string.
            Default - empty string.

    Returns:
        None.
    """
    ...
```

Функция как объект

```
def do_something() -> None:
    ...

>>> print(type(do_something).__name__)
function

>>> print(do_something.__name__)
do_something

>>> print(do_something.__doc__)
None

>>> print(do_something.__annotations__)
{'return': None}
```

Функции и переменные

```
def do_something() -> None:
    print('do something')

>>> my_var = do_something
>>> do_something()
>>> my_var()
do something
do something
```

Функции и коллекции

```
import math

functions = {
    math.sin: math.asin, math.sinh: math.asinh,
    math.cos: math.acos, math.cosh: math.acosh
}

for key, value in list(functions.items()):
    functions[value] = key

for key, value in functions.items():
    print(f'func: {key.__name__}; inverse func: {value.__name__}')
```

Функции как аргументы

```
from typing import Callable

def do_something() -> None:
    print('do something')


def do_something_with_func(func: Callable) -> None:
    print(f'do something with func: {func.__name__}')
    func()

>>> do_something_with_func(do_something)
do something with func: do_something
do something
```

Функции как результат

```
from typing import Callable

def produce_func() -> Callable:
    def do_something() -> None:
        print('produced function')

    return do_something

>>> produced_func = produce_func()
>>> print(f'type: {type(produced_func).__name__};')
>>> produced_func()
type: function;
produced function
```

Семинар