

Углубленный Python

Лекция 2 Функции

Кандауров Геннадий



Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

vk образование

БлогиЛюдиПрограммаВакансииРасписание

Q<

VK

Техно

Открыт приём заявок!

чт, 8 сентября

Нет занятий

пт, 9 сентября

18:00 Углубленный Py... с3
Введение в Python, основные
понятия, тестирование
Г. Кандауров

сб, 10 сентября

Нет занятий

вс, 11 сентября

Нет занятий

пн, 12 сентября

Нет занятий

Углубленный Python

↓ 0 ↑

Блог для материалов по курсу "Углубленный Python"

57 читателей, 2 топика

ПодписатьсяСоздать топик

Поиск по авторам, заголовку и тексту топика...

Найти

Добро пожаловать на курс!

Углубленный Python

ИзменитьУдалить

Всем привет и добро пожаловать на курс по углубленному изучению Python!

Прямой эфир

МоиВсе

Геннадий Кандауров час назад
Углубленный Python → Добро пожаловать
на курс! 0

Екатерина Черкасова 7 дней назад
Стажировка → Приглашаем мобильных,
фронтенд- и бэкэнд-разработчиков на
Weekend Offer! 0

Дарья Вовченко 9 дней назад
Углубленный Python → Добро пожаловать
в образовательные проекты VK
Образование! 0

Дарья Вовченко 9 дней назад
Разработка веб-сервисов на

Квиз про прошлой лекции



Содержание занятия

1. Функции, λ -функции
2. Параметры и аргументы
3. Декораторы
4. Пространства имен, LEGB
5. Встроенные функции

Функции



Функции

Функция — это именованный блок кода, который выполняет определенную задачу и может быть вызван из других частей программы.

Функции помогают организовать код, делая его более модульным, читаемым и повторно используемым.

- Чистые функции и с побочными эффектами
- Рекурсивные
- lambda-функции

Функции: именование

```
def square(x):  
    return x * x
```

```
>>> square(4)  
16
```

Правила наименования:

- имя функции состоит из букв, чисел, знака подчеркивания (_);
- название функции не должно начинаться с цифры;
- лидирующий знак подчеркивания - соглашение, что функция непубличная.

Функции: параметры и аргументы

- Позиционные аргументы

```
def count_params(a, b):...  
count_params(10, 20)
```

- Именованные аргументы

```
def count_params(a, b):...  
count_params(a=10, b=20)
```

- Параметры со значениями по умолчанию

```
def count_params(a, b=20):... # аккуратно со значением (!)  
count_params(10)
```


Функции: параметры и аргументы

- Произвольное количество позиционных аргументов

```
def count_params(a, *args):...  
count_params(10, 20, 30, 40)
```

- Произвольное количество именованных аргументов

```
def count_params(a, **kwargs):...  
count_params(10, b=20, c=30, d=40)
```

- Позиционные только параметры

```
def count_params(a, /):...  
count_params(10)  # count_params(a=10) - ?
```

- Именованные только параметры

```
def count_params(*, a, b=20):...  
count_params(a=10, b=20)  # count_params(10) - ?
```

Функции: параметры

```
def fn1(x, y=100): pass
```

```
def fn2(*args): pass
```

```
def fn3(**kwargs): pass
```

```
def fn4(*args, **kwargs): pass
```

```
def fn5(*, val): pass
```

```
def fn6(start, stop, /): pass
```

```
def fn7(pos1, /, pos2, pos3=3, *, kw1=11, **kwargs): pass
```

λ-функции

```
>>> min(list_of_objects, key=lambda obj: obj.x * obj.y)  
obj
```

- Содержат только одно выражение;
- Полезны в случае, когда нужна одноразовая функция;
- Работают потенциально быстрее классических функций;
- Потенциально повышают читаемость кода, но могут понизить.

Функции, lambda-функции

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

```
def do_action(action, *args, **kwargs):  
    print(f"do {action} with {args}, {kwargs}")  
  
    if callable(action):  
        return action(*args, **kwargs)  
    else:  
        return action
```

```
do_action(add, 1, b=2)  
do_action(lambda x, y: x * y, 5, y=6)
```

Функции: замыкание

Замыкание - функция первого класса, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её параметрами.

Функция, которая "запоминает" значения переменных из своей объемлющей области видимости, даже если эта область видимости больше не существует. Замыкания позволяют функциям сохранять состояние между вызовами и обеспечивают доступ к переменным, которые были в области видимости при создании функции, даже после завершения выполнения этой области.

```
def get_summator(a):  
    def add(b):  
        return a + b  
    return add
```

```
add2 = get_summator(2)  
add42 = get_summator(42)  
add2(5)    # 7  
add42(5)   # 47
```

Функции: замыкание

```
def make_function(name, *args, kw=12, **kwargs):  
    def inner(age=999):  
        print(f"{name=}, {age=}, {kw=}, {args=}, {kwargs=}")  
    return inner  
  
fn = make_function('skynet', 54, aim='term')  
fn()  
# name='skynet', age=999, kw=12, args=(54,), kwargs={'aim': 'term'}
```

Функции: декораторы

Декоратор - это функция, которая принимает функцию и возвращает новую функцию (возможно, с добавленной функциональностью).

```
def deco(fn):  
    def inner(*args, **kwargs):  
        print("before", fn.__name__)  
        res = fn(*args, **kwargs)  
        print("after", fn.__name__)  
        return res  
    return inner
```

```
@deco  
def add_nums(a, b):  
    return a + b
```

Функции: декораторы с параметрами

```
def deco(add_param):  
    def inner_deco(fn):  
        def inner(*args, **kwargs):  
            return fn(*args, **kwargs) + add_param  
        return inner  
    return inner_deco
```

```
@deco(45)  
def add_nums(a, b):  
    return a + b
```


Итераторы

Итератор представляет собой объект-перечислитель, который для данного объекта выдает следующий элемент, либо вызывает исключение, если элементов больше нет.

```
num_list = [1, 2, 3]
itr = iter(num_list)
print(next(itr))  # 1
print(next(itr))  # 2
print(next(itr))  # 3
print(next(itr))  # StopIteration
```

```
class SpecialIterator:
    def __init__(self, limit):
        self.limit = limit
        self.counter = 0
    def __iter__(self): return self
    def __next__(self):
        if self.counter < self.limit:
            self.counter += 1
            return self.counter
        else:
            raise StopIteration

itr_obj = SpecialIterator(3)
print(next(itr_obj))
```

Генераторы

Генератор – подпрограмма, которая может возвращать очередное значение и автоматически сохранять и возобновлять своё состояние для возврата следующего значения;
объект, который при вызове `next()` возвращает следующий элемент по алгоритму его работы.

```
def gen(count):  
    while count > 0:  
        yield count  
        count -= 1  
    return count # будет аргументом StopIteration  
  
for i in gen(5):  
    print(i) # 5, 4, 3, 2, 1
```

Функции как объекты

```
>>> fn.__dict__
```

```
{}
```

```
>>> fn.music = 'yes'
```

```
>>> fn.__dict__
```

```
{'music': 'yes'}
```

```
>>> fn.music
```

```
'yes'
```

Функции: атрибуты

__doc__ докстринг, изменяемое

```
>>> make_function.__doc__  
'makes inner function'
```

__name__ имя функции, изменяемое

```
>>> make_function.__name__  
'make_function'
```

__qualname__ fully qualified имя, изменяемое

```
>>> make_function.__qualname__  
'make_function'
```

```
>>> fn.__qualname__
```

```
'make_function.<locals>.inner'
```

Функции: атрибуты

`__defaults__` кортеж дефолтных значений, изменяемое

```
>>> fn.__defaults__  
(999,)
```

`__kwdefaults__` словарь дефолтных значений кваргов, изменяемое

```
>>> make_function.__kwdefaults__  
{'kw': 12}
```

`__closure__` кортеж свободных переменных функции

```
>>> make_function.__closure__  
None  
>>> fn.__closure__[0].cell_contents  
(54,)
```

Пространства имен

*“Namespaces are one honking great
idea -- let's do more of those!”*

Tim Peters (import this)



Пространства имен

Пространство имен—это совокупность определенных в настоящий момент символических имен и информации об объектах, на которые они ссылаются.

- Встроенное
- Глобальное
- Объемлющее
- Локальное

Область видимости: LEGB

Область видимости имени это часть программы, в которой данное имя обладает значением.

Интерпретатор определяет эту область в среде выполнения, основываясь на том, где располагается определение имени и из какого места в коде на него ссылаются.

- | | |
|---------------|--------------------------|
| 1. Локальная | ○ <code>globals()</code> |
| 2. Объемлющая | ○ <code>locals()</code> |
| 3. Глобальная | ○ <code>global</code> |
| 4. Встроенная | ○ <code>nonlocal</code> |

__builtins__

```
>>> hasattr(__builtins__, "dir")
```

```
True
```

```
>>> dir(__builtins__)
```

```
...
```

int	map	reversed	dir
float	zip	len	type
str	filter	sum	isinstance
bool	range	all	issubclass
tuple	enumerate	any	hasattr
list	sorted	globals	getattr
dict	min	locals	setattr
set	max	callable	delattr

Еще немного про тесты

All unit tests passing.



ALT

Домашнее задание #02

- Обработчик json
- Декоратор с повторными запусками
- +тесты
- flake8 + pylint перед сдачей
- Зеленый пайплайн



Напоминание отметиться на портале Vol 2

+ ОСТАВИТЬ ОТЗЫВ ПОСЛЕ ЛЕКЦИИ

vk образование

БлогиЛюдиПрограммаВакансииРасписание

Q<

VK

Техно

Открыт приём заявок!

чт, 8 сентября

Нет занятий

пт, 9 сентября

18:00 Углубленный Py... с3
Введение в Python, основные
понятия, тестирование
Г. Кандауров

сб, 10 сентября

Нет занятий

вс, 11 сентября

Нет занятий

пн, 12 сентября

Нет занятий

Углубленный Python

↓ 0 ↑

Блог для материалов по курсу "Углубленный Python"

57 читателей, 2 топика

Подписаться

Создать топик

Поиск по авторам, заголовку и тексту топика...

Найти

Добро пожаловать на курс!

Углубленный Python

Изменить

Удалить

Всем привет и добро пожаловать на курс по углубленному изучению Python!

Прямой эфир

МоиВсе

Геннадий Кандауров час назад
Углубленный Python → Добро пожаловать
на курс! 0

Екатерина Черкасова 7 дней назад
Стажировка → Приглашаем мобильных,
фронтенд- и бэкэнд-разработчиков на
Weekend Offer! 0

Дарья Вовченко 9 дней назад
Углубленный Python → Добро пожаловать
в образовательные проекты VK
Образование! 0

Дарья Вовченко 9 дней назад
Разработка веб-сервисов на

Спасибо за
внимание

