PYTHON

Лекция 6

ПЛАН ЛЕКЦИИ

- Функции: определение и вызов
- Параметры функций
- Введение в декораторы

СТРУКТУРА ОПРЕДЕЛЕНИЯ

- Ключевое слово def
- Имя
- Параметры
- Тело

ПАРАМЕТРЫ

- Задаются в определении как перечень имен
- Привязка в точке вызова
- Можно по позиции, можно по имени
- Но можно ограничить способ задания параметра

ПРИМЕР

```
1 def sum(a, b):
2     return a + b
3
4 print(sum(1, 2))
5 print(sum(a=1, b=2))
6 #print(sum(a=1, 2))
7 print(sum(b=1, a=2))
8 #print(sum(b=1, 2))
```

TRADEOFF

- Плюс именованных параметров - помогают не перепутать параметры
- Повышают наглядность кода
- Минус: становятся частью интерфейса
- Разнобой в традициях именования

ЧТО ПРЕДЛАГАЕТ PYTHON

- В момент определения функции можно выбрать стиль параметров
- Можно разный для разных параметров
- Вариант 1: чисто позиционные параметры
- Вариант 2: чисто именованные параметры
- Вариант 3: смешанные параметры

ЧТО ПРЕДЛАГАЕТ PYTHON

- В базовом варианте определения все параметры смешанные
- Можно в конце указать псевдопараметр /
- Тогда все параметры будут позиционными
- Или посередине
- Кто левее - позиционные, кто правее - смешанные

ПРИМЕР

```
1 def sum(a, b, /):
2     return a + b
3
4 print(sum(1, 2))
5 #print(sum(a=1, b=2))
6 #print(sum(a=1, 2))
7 print(sum(b=1, a=2))
8 ##print(sum(b=1, 2))
```

ПРИМЕР

```
1 def is_near(a, b, /, epsilon):
2     return abs(a - b) < epsilon
3
4 print(is_near(1, 1.1, 0.01))
5 print(is_near(1, 1.0001, 0.01))
6 print(is_near(1, 1.0001, epsilon=0.01))
```

ЧТО ПРЕДЛАГАЕТ PYTHON

- Можно в начале указать псевдопараметр %
- Тогда все параметры будут именованными
- Или посередине
- Кто левее - смешанные, кто правее - именованные

ПРИМЕР

```
1 def find_car(*, year, mileage):
2     pass
3
4 #find_car(1995, 50000)
5 find_car(year=1995, mileage=50000)
6 find_car(mileage=50000, year=1995)
```

ПРИМЕР

```
1 def update_car(id, *, year, mileage):
2     pass
3
4 #update_car('qwertyuio', 1995, 50000)
5 find_car('qwertyuio', year=1995, mileage=50000)
6 find_car('qwertyuio', mileage=50000, year=1995)
7 find_car(id='qwertyuio', year=1995, mileage=50000)
8 find_car(id='qwertyuio', mileage=50000, year=1995)
```

В ОБЩЕМ ВИДЕ

- Можно использовать оба знака сразу
- Отсутствие знаков - это как будто начинается с % и заканчивается *
- Можно поставить рядом / и *
- Тогда не будет смешанных параметров

ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

- Параметру можно указывать значение по умолчанию
- Это не зависит от того, какой у него стиль
- Указываем значение по умолчанию через знак равенства
- Для позиционных и смешанных - параметры со значением по умолчанию идут "в конце"

ПРИМЕР

```
1 def sum(a, b=0, /):
2     return a + b
3
4 print(sum(1, 2))
5 print(sum(1))
```

ПРИМЕР

```
1 def is_near(a, b=0, /, epsilon=0.0001):
2     return abs(a - b) < epsilon
3
4 print(is_near(1))
5 print(is_near(1, 1.1))
6 print(is_near(1, 1.1, 0.01))
7 print(is_near(1, 1.0001, epsilon=0.01))
8 print(is_near(1, epsilon=0.01))
```

ПРИМЕР

```
1 def find_car(*, year=2000, mileage=30000):
2     pass
3
4 find_car(year=1995, mileage=50000)
5 find_car(mileage=50000, year=1995)
6 find_car(mileage=50000)
7 find_car(year=1995)
8 find_car()
```

ПРИМЕР

```
1 def update_car(id='0', *, year, mileage):
2     pass
3
4 update_car('qwertyuio', year=1995, mileage=50000)
5 update_car(mileage=50000, year=1995)
6 update_car(year=1995, mileage=50000, id='qwertyuio')
7 update_car(id='qwertyuio', mileage=50000, year=1995)
```

ПЕРЕМЕННОЕ КОЛИЧЕСТВО ПАРАМЕТРОВ

- Иногда мы сразу не знаем, сколько параметров будет
- Как для позиционных, так и для именованных
- Например, в функции типа max
- В одном месте хотим от пяти аргументов, в другом от трех

ПЕРЕМЕННОЕ КОЛИЧЕСТВО ПАРАМЕТРОВ

- Или мы в вызове указываем свойства объекта
- В качестве именованных параметров
- Какие-то могут оказаться известными заранее, а какие-то - нет

ПЕРЕМЕННОЕ КОЛИЧЕСТВО ПАРАМЕТРОВ

- Перед параметром в описании функции может стоять * или **
- * - параметр принимает в себя переменное число позиционных параметров в точке вызова
- ** - переменное число именованных параметров
- * - внутри превращается в кортеж, ** - в словарь

ПРИМЕР

```
 1 def f(*args):
 2     print(type(args))
 3     print(len(args))
 4     for i, v in enumerate(args):
 5         print(i, v)
 6
 7 f()
 8 f(1)
 9 f(1, 2)
10 #f(1, 2, n=3) # так нельзя
```

ПРИМЕР

```
 1 def f(a, *args):
 2     print(a)
 3     print(type(args))
 4     print(len(args))
 5     for i, v in enumerate(args):
 6         print(i, v)
 7
 8 #f() - нельзя
 9 f(1)
10 f(a=1)
11 f(1, 2)
12 #f(a=1, 2) # так нельзя
```

ПРИМЕР

```
 1 def f(a=1, *args):
 2     print(a)
 3     print(type(args))
 4     print(len(args))
 5     for i, v in enumerate(args):
 6         print(i, v)
 7
 8 f()
 9 f(a=22)
10 f(22)
11 f(33, 44)
12 # f(a=33, 44)
```

СТРАННОСТИ

- *-параметр не любит псевдопараметров
- В чем-то это логично
- Местами - не совсем
- Например, нельзя так:

```
def f(*args, /, *, name):
```

ПРИМЕР

```
 1 def f(*args, name):
 2     print(name)
 3     print(type(args))
 4     print(len(args))
 5     for i, v in enumerate(args):
 6         print(i, v)
 7
 8 f(name=1)
 9 f(77, name=1)
10 f(1, 2, name=1)
```

ЛОГИКА PYTHON

- *args сам по себе является границей перед строго именованными параметрами
- * - особый случай
- После параметра со звездочкой начинаются именованные параметры
- Поэтому и *args перед / запрещен

ПРИМЕР

```
1 def f(**kwargs):
2     print(type(kwargs))
3     print(kwargs)
4
5 f()
6 f(name=1)
```

ПРИМЕР

```
1 def f(name, **kwargs):
2     print(type(kwargs))
3     print(kwargs)
4
5 f(123)
6 f(name=1)
7 f(234, name=1)
```

ПРИМЕР

```
1 def f(name=111, **kwargs):
2     print(type(kwargs))
3     print(kwargs)
4
5 f()
6 f(123)
7 f(name=1)
8 f(234, name=1)
```

ЛОГИКА PYTHON

- **kwargs - тоже со звездочкой
- Но с просто звездочкой сочетается
- С / - тоже сочетается, если идет справа
- После **kwargs никакие параметры идти не могут

ПРИМЕР

```
1 def f(name=111, /, **kwargs):
2     print(name)
3     print(type(kwargs))
4     print(kwargs)
5
6 f()
7 f(123)
8 f(1, k=2, name=2)
9 f(234, name=1)
```

ПРИМЕР

```
1 def f(a=1, /, b=2, *, c, **kwargs):
2     print(type(kwargs))
3     print(kwargs)
4
5 f(c=1)
6 f(123, c=3, b=5, d=123)
```

ОСОБЫЙ ПРИМЕР

```
1 def f(*args, **kwargs):
2     print(args)
3     print(kwargs)
4
5 f()
6 f(c=1)
7 f(123, c=3, b=5, d=123)
8 f(123, 234, c=3, b=5, d=123)
```

ДЕКОРИРОВАНИЕ

- Это еще не Python-декораторы
- Но шаг в ту сторону
- Хотим взять функцию и ее вызов сопроводить какими-то действиями
- Например: напечать сам факт вызова и параметры
- Можно считать вызовы, кешировать результаты, делать повторный вызов и т.п.

ДЕКОРИРОВАНИЕ

- Можем написать функцию, которая примет в качестве параметра другую функцию
- И ее параметры
- Внутри себя сделает нужные действия и вызовет функцию
- Но если "основная" функция - произвольная, то и параметры могут быть произвольными
- Тут помогут *args и **kwargs

ПРИМЕР

```
1 def f1(a, b, /, *, c, d):
2     return a + b + c + d
3
4 def f2(a):
5     return a * 2
6
7 def f3(**kwargs):
8     return len(kwargs)
```

ПРИМЕР

```
1 def trace(func, *args, **kwargs):
2     print(func)
3     print(args)
4     print(kwargs)
5
6 trace(f1, 1, 2, c=3, d=4)
7 trace(f2, 5)
8 trace(f3, q=6, w=7, e=8)
```

ПОКА НЕ ВСЕ

- То, что мы держим в переменных args/kwargs, надо передать в функцию
- Нужна какая-то особая конструкция - и она есть
- Если есть последовательность значений, ее можно поэлементно превратить в параметры
- В нашем случае последовательность - args

ПОКА НЕ ВСЕ

- Конструкция выглядит так: *args
- Похоже на то, что в определении функции
- Но тут другой контекст и другой смысл
- По сути - что-то вроде "обратной операции" над той же переменной
- **kwargs - аналогично

ПРИМЕР

```
1 def f1(a, b, /, *, c, d):
2     return a + b + c + d
3
4 def f2(a):
5     return a * 2
6
7 def f3(**kwargs):
8     return len(kwargs)
```

ПРИМЕР

```
1 def trace(func, *args, **kwargs):
2     print(func)
3     print(args)
4     print(kwargs)
5     return func(*args, **kwargs)
6
7 trace(f1, 1, 2, c=3, d=4)
8 trace(f2, 5)
9 trace(f3, q=6, w=7, e=8)
```

ИЗМЕНИМ ПОДХОД

- Передавали в функцию сразу функцию и параметры
- А давайте передадим только функцию
- И вернем декорированную функцию
- В которую можно передавать параметры

ПРИМЕР

```
1 def f1(a, b, /, *, c, d):
2     return a + b + c + d
3
4 def f2(a):
5     return a * 2
6
7 def f3(**kwargs):
8     return len(kwargs)
```

ПРИМЕР

```
 1 def trace(func):
 2     def f(*args, **kwargs):
 3         print(func)
 4         print(args)
 5         print(kwargs)
 6         return func(*args, **kwargs)
 7     return f
 8
 9 trace(f1)(1, 2, c=3, d=4)
10 trace(f2)(5)
11 trace(f3)(q=6, w=7, e=8)
```

РАЗОВЬЕМ ИДЕЮ

```
1 def f1(a, b, /, *, c, d):
2     return a + b + c + d
3
4 def f2(a):
5     return a * 2
6
7 def f3(**kwargs):
8     return len(kwargs)
```

РАЗОВЬЕМ ИДЕЮ

```
 1 def trace(func):
 2     def f(*args, **kwargs):
 3         print(func.__name__)
 4         print(args)
 5         print(kwargs)
 6         return func(*args, **kwargs)
 7     return f
 8
 9 f1 = trace(f1)
10 f2 = trace(f2)
11 f3 = trace(f3)
12 f1(1, 2, c=3, d=4)
13 f2(5)
14 f3(q=6, w=7, e=8)
```

НЕЗАВИСИМОСТЬ МЕХАНИЗМОВ

- Переменное число параметров и подстановка списка/словаря - два отдельных механизма
- Можно пару значений, хранящуюся в переменных, подставить на место двух обычных параметров
- И вернем декорированную функцию
- В которую можно передавать параметры

ПРИМЕР

```
1 def f(a, b, c, *args):
2     print(a, b, c, args)
3
4 f(*'vasya')
5 f(*'vasya', 'petya')
6 f(*'vasya', 'petya', *'dima')
```

ПРИМЕР

```
1 def f(a, b, c, *args, **kwargs):
2     print(a, b, c, args)
3     print(kwargs)
4
5 f(*'vasya', 'petya', *'dima', *{'c': 22, 'kolya': 33})
6 f(*'vasya', 'petya', *'dima', **{'c': 22, 'kolya': 33})
```

ДЕКОРАТОР

```
 1 def trace(func):
 2     def f(*args, **kwargs):
 3         print(func.__name__)
 4         print(args)
 5         print(kwargs)
 6         return func(*args, **kwargs)
 7     return f
 8
 9 @trace
10 def f1(a, b, /, *, c, d):
11     return a + b + c + d
```

ДЕКОРАТОР

```
 1 @trace
 2 def f2(a):
 3     return a * 2
 4
 5 @trace
 6 def f3(**kwargs):
 7     return len(kwargs)
 8
 9
10 f1(1, 2, c=3, d=4)
11 f2(5)
12 f3(q=6, w=7, e=8)
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

- Функция в Python всегда что-то возвращает
- Либо то, что указано в return
- Если ничего не указано, то None
- Если нет return, то тоже None