eebb

ЯЗЫК PYTHON

Лекция 7. Функции. Часть 1

Что такое функция

- Переиспользуемый именованный фрагмент кода
- Со списком параметров
- Возможно, пустым
- И возвращаемым значением

Синтаксис

- Начинаем с ключевого слова def
- Дальше - имя функции
- Список параметров в скобках
- Двоеточие
- Это все - заголовок функции

Синтаксис

- Дальше - тело
- Со стандартным отступом от заголовка
- Блок кода
- С использованием параметров в качестве переменных
- И return для возвращения результата

Пример функций без параметров

```
 1 def hello():
 2     print('Hello')
 3
 4
 5 def hello_many():
 6     for _ in range(10):
 7         print('Hello')
 8
 9
10 hello()
11 hello()
12 hello_many()
```

Пример функций с параметрами

```
1 def hello_many(n):
2     for _ in range(n):
3         print('Hello')
4
5 def hello(name):
6     print('Hello, ' + name)
7
8 hello_many(4)
9 hello('vasya')
```

Определение параметров

- В простейшем варианте - список имен переменных
- Тогда при вызове функции количество выражений в скобках должно равняться количеству переменных в определении
- При вызове функции все упоминания имен параметров в теле
- Заменяются на значения, используемые в точке вызова

Возвращаемое значение

- Вызов функции может стоять в том месте, где требуется значение
- Например, в качестве параметра функции (хотя бы print)
- И в качестве операнда в выражении
- Тогда при вычислении возвращаемое значение заменяет вызов функции

Возвращаемое значение

- Если функция при исполнении доходит до конца тела, не встретив return
  - Функция возвращает None
- Если функция при исполнении доходит до return без выражения
  - Функция возвращает None
- Если функция при исполнении доходит до return с выражением
  - Функция возвращает результат вычисления выражения

Пример на возвращение значений

```
 1 def incr(n):
 2     return n + 1
 3
 4 def hello(name):
 5     print('Hello, ' + name)
 6
 7 def f():
 8     return
 9
10 print(incr(5)) # 6
11 print(hello('vasya')) # None
12 print(f()) # None
```

Значения параметров по умолчанию

- Допустим, хотим реализовать функцию incr
- Увеличивать значение переменной
- На произвольное значение
- Но чаще всего мы будем увеличивать на 1

Значения параметров по умолчанию

- Обобщая - у нас есть функция с параметрами
- У некоторых параметров есть особо частотные значения
- И хочется их не упоминать вообще, если нам нужно частное значение
- А упоминать - только когда нужно другое

Пример на значение по умолчанию

```
1 def incr(n, d=1):
2     return n + d
3
4 print(incr(5)) # 6
5 print(incr(5, 1)) # 6
6 print(incr(5, 10)) # 15
7 print(incr(5, -2)) # 3
```

Коварный момент

- Допустим, есть функция с двумя параметрами
- Первый параметр - строка
- Второй - список из 6 элементов
- Хотим написать функцию, которая считает количество гласных в строке
- Точнее, каждой из гласных

Коварный момент

- Хотим накопительного эффекта
- Поэтому передаем второй параметр
- В котором могут быть начальные значения

Пример на коварный момент. Первый шаг

```
 1 def count_vowels(text, counts):
 2     vowels = 'aeiouy'
 3     for c in text.lower():
 4         index = vowels.find(c)
 5         if index >= 0:
 6             counts[index] += 1
 7
 8 counts = [0] * 6
 9 count_vowels('hello', counts)
10 print(counts)
11 count_vowels('index', counts)
12 print(counts)
```

Коварный момент

- А теперь мы хотим вызывать count_vowels в цикле
- И печатать промежуточные результаты
- И хотим укоротить код
- Давайте еще возвращать значение counts

Пример на коварный момент. Второй шаг

```
 1 def count_vowels(text, counts):
 2     vowels = 'aeiouy'
 3     for c in text.lower():
 4         index = vowels.find(c)
 5         if index >= 0:
 6             counts[index] += 1
 7     return counts
 8
 9 counts = [0] * 6
10 print(count_vowels('hello', counts))
11 print(count_vowels('index', counts))
```

Коварный момент

- Получили функцию с несколькими сценариями использования
- И обновляет переданный список
- И возвращает обновленное значение
- Можем опираться на любой вариант, по ситуации

Коварный момент

- И тут нам захотелось добавить еще один сценарий
- Допустим, нам хочется просто посчитать количество гласных один раз
- И вернуть значение
- Используем параметр по умолчанию

Пример на коварный момент. Третий шаг

```
 1 def count_vowels(text, counts=[0, 0, 0, 0, 0, 0]):
 2     vowels = 'aeiouy'
 3     for c in text.lower():
 4         index = vowels.find(c)
 5         if index >= 0:
 6             counts[index] += 1
 7     return counts
 8
 9 print(count_vowels('hello'))
10 print(count_vowels('hello')) # тут будет неожиданность
```

Как работают параметры по умолчанию

- Все параметры по умолчанию вычисляются в момент определения функции
- Полученные значения сохраняются вместе с функцией
- Они НЕ перевычисляются заново
- Для неизменяемых типов это неважно

Как работают параметры по умолчанию

- Для изменяемых - опасно
- Любое изменение параметра со значением по умолчанию повлияет на последующие вызовы
- Популярный способ избежать проблемы - использовать None как значение по умолчанию
- И использовать if в начале тела

Пример на коварный момент. Исправленная версия

```
 1 def count_vowels(text, counts=None):
 2     if counts is None:
 3         counts = [0, 0, 0, 0, 0, 0]
 4     vowels = 'aeiouy'
 5     for c in text.lower():
 6         index = vowels.find(c)
 7         if index ≥ 0:
 8             counts[index] += 1
 9     return counts
10 
11 print(count_vowels('hello'))
12 print(count_vowels('hello'))
```

Позиционные и именованные параметры

- Бывают функции, у которых параметров много
- И "естественного" порядка нет
- Например, если параметры - разные атрибуты одного понятия
- Тогда легко запутаться в момент вызова

Позиционные и именованные параметры

- Пусть надо сохранить данные о заказе
- Определяем функцию save_order(order_id, client_id, order_date, total_price):
- В точке вызова легко перепутать порядок
- Дату заказа поменять местами с id клиента

Позиционные и именованные параметры

- Python позволяет использовать именованные параметры
- При вызове функции указывать имя параметра
- И не придерживаться порядка параметров
- При задании их значения

Позиционные и именованные параметры

- То, как мы задавали параметры в уже бывших примерах определений
- Позволяет задавать их в точке вызова и позиционно, и по имени
- Но иногда именованность вредна
- Например, если функция - математическая
- И нет устоявшегося имени параметра (sin, log, и т.п.)

Чисто позиционные параметры

- Чисто позиционные параметры указываются в начале определения функции
- И отделяются от остальных псевдопараметром /
- Если других нет, то в конце списка параметров идет /

Пример на чисто позиционные параметры

```
1 def incr(v, /):
2     return v + 1
3
4 print(incr(5))
5 print(incr(v=5))  # некорректно
```

Пример на несколько чисто позиционных

```
1 def incr(v, delta, /):
2     return v + delta
3
4 print(incr(5, 3))
5 print(incr(v=5, delta=3))  # некорректно
6 print(incr(5, delta=3))  # некорректно
```

Пример на чисто позиционные с обычными

```
1 def incr(v, /, delta):
2     return v + delta
3
4 print(incr(5, 3))
5 print(incr(v=5, delta=3))  # некорректно
6 print(incr(5, delta=3))
```

Чисто именованные параметры

- Иногда хочется обязать использовать именованные параметры при вызове
- Такие параметры ставятся в конец списка при определении
- И отделяются от других псевдопараметром *
- Если других нет, список начинается с *

Пример на чисто именованные

```
1 def wiki_url(*, lang, name):
2     return "https://{0}.wikipedia.org/wiki/{1}".format(lang,
3
4 print(wiki_url(lang='en', name='Python_(programming_language
5 print(wiki_url(lang='en', 'Python_(programming_language)'))
6 print(wiki_url('en', name='Python_(programming_language)'))
7 print(wiki_url('en', 'Python_(programming_language)'))
```

Список параметров в общем виде

- Сначала чисто позиционные
- Затем /
- Затем универсальные
- Затем *
- Затем чисто именованные

Список параметров в общем виде

- Если универсальных нет
- А позиционные и именованные - есть
- Универсальные опускаем
- Все остальное - на своих местах

Список параметров в общем виде

- Если нет только позиционных
  - Универсальные, *, именованные
- Если нет только именованных
  - Позиционные, / универсальные

Список параметров в общем виде

- Если только позиционные
  - Позиционные, /
- Если только именованные
  - *, именованные
- Если только универсальные - просто без знаков

Параметры со значением по умолчанию

- Осмыслим это явление в свете трех групп параметров
- Не должно быть двусмысленности
- Например, если параметры чисто позиционны
  - То все параметры со значением по умолчанию должны идти в конце

Почему в конце

- Если не в конце: def f(a, b=5, c=6, d, /): ....
- Непонятно, как интерпретировать некоторые вызовы
- Например: f(1, 2, 3)
- Больше одной интерпретации

Параметры со значением по умолчанию

- А для чисто именованных это не имеет значения
- Потому что все именем определяется
- def f(*, a, b=5, c=6, d): ....
- Вызов: f(a=1, b=2, d=3)
- Или: f(a=1, c=2, d=3)

Правила для вызова

- Именованные идут за позиционными
- Именованным должно соответствовать имя параметра в определении
- Порядок именованных не важен
- Не должно оставаться параметров без значений
- Не должно быть двойного присваивания параметру

Пример с обычными параметрами

```
1 def f(a, b, c):
2     print(a, b, c)
3 
4 f(1, 2, 3)
5 f(1, 2, c=3)
6 f(1, b=2, c=3)
7 f(1, c=2, b=3)
8 f(a=1, c=2, b=3)
9 f(c=1, b=2, a=3)
```

Примеры некорректных вызовов

```
1 def f(a, b, c):
2     print(a, b, c)
3
4 f(1, 2)              # c не определено
5 f(1, c=3)            # b не определено
6 f(b=2, c=3)          # a не определено
7 f(1, c=2, b=3, d=4) # несуществующий именованный
8 f(1, 2, 3, 4)        # лишний позиционный
9 f(1, c=1, b=2, a=3) # двойное присваивание a
```

Пример: обычные параметры, значения по умолчанию

```
1 def f(a, b=100):
2     print(a, b)
3
4 f(1, 2)
5 f(1)
6 f(a=1)
7 f(b=2, a=3)
8 f(a=1, b=3)
```

Пример: обычные параметры, значения по умолчанию

```
1 def f(a, b=100):
2     print(a, b)
3
4 f(b=1, 2)          # именованный до позиционного
5 f(b=1)             # a не определено
6 f(10, a=1)         # a дважды определено
```

Пример: строго позиционные параметры

```
1 def f(a=20, b=50, c=100, /):
2     print(a + b + c)
3
4 f()
5 f(1)
6 f(1, 2)
7 f(1, 2, 3)
```

Пример: строго позиционные и смешанные параметры

```
1 def f(a=20, b=50, /, c=100):
2     print(a + b + c)
3
4 f()
5 f(1)
6 f(1, 2)
7 f(1, c=2)
8 f(1, 2, 3)
9 f(1, 2, c=3)
```

Пример: строго позиционные и строго именованные

```
1 def f(a=20, b=50, /, *, c=100):
2     print(a + b + c)
3
4 f()
5 f(1)
6 f(1, 2)
7 f(1, c=2)
8 f(1, 2, c=3)
```