Основы Python

Функции, модули

- синтаксис определения функций
- определение и исполнение функций
- области видимости переменных
- импортирование переменных и функций из других файлов
- понятие модульности

Синтаксис объявления функций

- ключевое слово def и идентификатор
- обязательные параметры
- не обязательные параметры и дефолтные значения
- передача параметров позиционно и по имени
- строка документации
- функция тоже объект
- возвращаемое значение

Ключевое слово def и имя функции

```
def hello():
    print “Hello world!”

hello()
```

- к имени функции требование те же что и к любому идентификатору: допускаются только буквы, цифры и знак подчеркивания
- по этому имени после определение функции можно вызывать функцию на исполнение

Обязательные параметры

```
def hello(name):
    print “Hello ” + name + “!”

hello(“world”)

n = “python”
hello(n)
```

- Если при определении функции был определен параметр, то при вызове функции, обязательно нужно передать параметр в функцию
- Можно передавать как константное значение так и переменную
- Внутрь функции переменная попадает именно с тем именем, что было в определении

Не обязательные параметры и дефолтные значения

```
def hello(name, surname='', title='Mr.'):
    if surname:
        surname = ' ' + surname.strip()
    print "Hello %s %s%s" % (title, name, surname)

hello("John")
hello("John", "Galt")
hello("John", "Galt", "Sir")
```

- Параметры могут иметь значения по умолчанию и тогда при вызове функции данные параметры указывать не обязательно
- Если параметры явно указать, то они затрут дефолтные значения

Передача параметров позиционно и по имени

```
def hello(name, surname='', title='Mr.'):
    if surname:
        surname = ' ' + surname.strip()
    print "Hello %s %s%s" % (title, name, surname)

hello("John")
hello("John", "Galt", "Sir")
hello("John", title="Sir")
hello("John", title="Sir", surname="Galt")
hello(title="Sir", surname="Galt", name="John")
```

- Передавая параметры в функцию они попадают в переменные по очереди
- Можно в явном виде указывать имена параметров которые вы передаете в функцию
- Вначале идут позиционные параметры, а потом именованые

Передача параметров позиционно и по имени

```
def hello(name, surname='', title='Mr.'):
    if surname:
        surname = ' ' + surname.strip()
    print "Hello %s %s%s" % (title, name, surname)

hello("John")
hello("John", "Galt", "Sir")
hello("John", title="Sir")
hello("John", title="Sir", surname="Galt")
hello(title="Sir", surname="Galt", name="John")
```

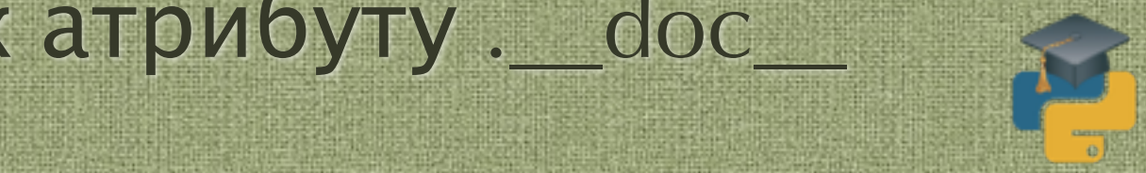
- Передавая параметры в функцию они попадают в переменные по очереди
- Можно в явном виде указывать имена параметров которые вы передаете в функцию
- Вначале идут позиционные параметры, а потом именованные

Строка документации

```
def hello(name, surname=‘’, title=‘Mr.’):
    “””
    There is function for greeting somebody
    “””
    if surname:
        surname = ‘ ’ + surname.strip()
    print “Hello %s %s%s” % (title, name,
surname)

print hello.__doc__
```

- Первой строкой после строки определения функции написав блочный коментарий, он будет строкой документации функции
- доступ к документации функции можно получить обратившись к атрибуту .__doc__

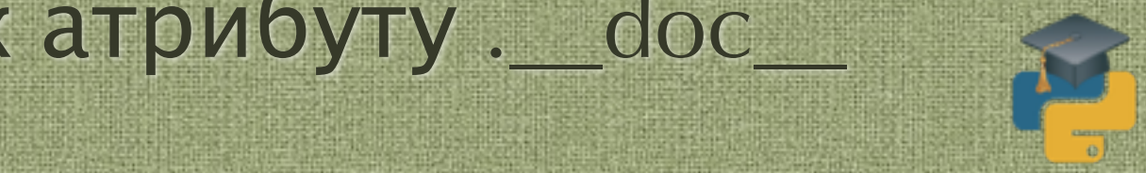


Функция тоже объект

```
def hello(name, surname='', title='Mr.'):
    """
    There is function for greeting somebody
    """
    if surname:
        surname = ' ' + surname.strip()
    print "Hello %s %s%s" % (title, name, surname)

print hello.__doc__
print hello.__name__
print dir(hello)
```

- Первой строкой после строки определения функции написав блочный коментарий, он будет строкой документации функции
- доступ к документации функции можно получить обратившись к атрибуту .__doc__



Возвращаемое значение

```
def some_function():
    pass

r = some_function() # r = None

def some_function():
    return 1

r = some_function() # r = 1
```

- Функция всегда по дефолту возвращает None
- С помощью return можно явно вернуть значение

Возвращаемое значение

```
def max_value(a, b):
    if a>b:
        return a
    elif a<b:
        return b
    return None

r = max_value(1, 2) # r = 2
r = max_value(4, 3) # r = 4
r = max_value(5, 5) # r = None
```

- выходов из функции может быть несколько, после выполнения инструкции return, функция сразу вернет значение и далее выполняться не будет

Определение и исполнение функций

```
def hello():
    print “Hello world!”

def zero_division():
    return 1/0

a = 1/0
```

- Т.к. язык не компилируемый, то функции интерпритируются только на момент вызова. И если там есть логические ошибки, то мы их обнаружим потом

Области видимости переменных

```
s = “Hello world”
def hello():
    print s
hello() # “Hello world”
s = “Good by!”
hello() # “Good by!”
def hello_local():
    s = “Hello again!”
    print s
hello_local() # “Hello again!”
s = “Globals are not good!”
hello() # “Globals are not good!”
hello_local() # “Hello again!”
```

- Функции создают локальное пространство переменных, но имеют доступ к глобальному

Области видимости переменных

```
s = “Hello world”
def hello(s):
    print s
hello(“Hello here!”) # “Hello here!”
print s # “Hello world”
hello(s) # “Hello world”
```

- В идеале вообще не использовать глобальных переменных
- ZEN: Namespaces are one honking great idea -- let's do more of those!

Области видимости переменных

```
l = [1, 2]
def increment_items(list_arg):
    for i in range(len(list_arg)):
        list_arg[i] += 1
    return list_arg
new_l = increment_items(l) # [2, 3]
print l # [2, 3]
def increment_items(list_arg):
    new_list = []
    for i in list_arg:
        new_list.append(i + 1)
    return new_list
new_l = increment_items(l) # [3, 4]
print l # [2, 3]
```

- Нужно быть осторожным при передаче в функцию изменяемых объектов
- При присваивании изменяемых объектов просто создаются новое имя для той же самой переменной

Разбор квадратного уравнения

```
def get_discr(a, b, c):
    d = b ** 2 - 4 * a * c
    return d

def get_eq_root(a, b, d, order=1):
    if order==1:
        x = (-b + d ** (1/2.0)) / 2*a
    else:
        x = (-b - d ** (1/2.0)) / 2*a
    return x
```

- Выделить отдельные функции и потом их использовать

Квадратное уравнения по модулям

- Перенести функции в отдельный файл
- Варианты импорта
- Альтернативный файл запуска функции

зачем же оно все-таки нужно __name__ == '__main__'

- Перенос файла с функциями во вложеный каталог
- Папка как модуль, роль __init__.py
- Точечная нотация, относительный импорт

lambda

```
def squared(x):
    return x**2

def task1(x):
    if type(x) is list:
        return [squared(i) for i in x]
    elif type(x) is tuple:
        return tuple(map(squared, x))

squared = lambda i: i**2
map(squared, x)
map(lambda i: i**2, x)
```

- безымянная функция
- просто альтернативный синтаксис
- можно использовать сразу при передаче функции как параметра

Распаковка параметров функции

```
def hello(name, surname='', title='Mr.'):
    if surname:
        surname = ' ' + surname.strip()
    print "Hello %s %s%s" % (title, name, surname)
a = ["John", "Galt"]
kwa = {surname:'Galt', title:'Sir'}
hello(*a)
hello("John", **kwa)
```

- *args - распаковка списка в позиционные переменные
- **kwargs - распаковка словаря в именованные переменные
- можно по-всякому комбинировать

Итого

- Узнали все про функции
- Области видимости переменных
- Определение и исполнение
- Структура и модульность