Программирование
на Python

Презентация занятия

Работа с библиотеками Python.

22 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2019

Тема: Работа с библиотеками Python.

Что хорошо, а что плохо?

1. Имена модулей и пакетов

А) import My-First-VKontakte-API-Modul

Б) import vkapi

2. Имена переменных

А) my_variable = 'Variable’

Б) My-Variable = 'Variable’

3. Имена классов

А) class my_first_class:

Б) class MyFirstClass:

Тема: Работа с библиотеками Python.

Что хорошо, а что плохо?

4. Пробелы и скобки

А) pineapple(pine[1], { apple: 2 })

Б) pineapple(pine[1], {apple: 2})

В) dish['ingredients'] = cook_book[:3]

Г) dish ['ingredients'] = cook_book [:3]

Д) if number_of_goods==4:

Е) if number_of_goods == 4:

Тема: Работа с библиотеками Python.

Что хорошо, а что плохо?

5. Отступы

А)
```
# Выровнено по открывающему разделителю
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

Б)
```
# Аргументы на первой линии запрещены, если не используется вертикальное
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```

6. Модули

А)
```
from subprocess import Popen, PIPE
```

Б)
```
import sys, os
```

В)
```
import os
import sys
```

Тема: Работа с библиотеками Python.

Что хорошо, а что плохо?

7. Сопоставьте стили

1. lowercase
2. lower_case_with_underscores
3. UPPERCASE
4. UPPERCASE_WITH_UNDERSCORES
5. CapitalizedWords

А) Имена модулей и пакетов
Б) Имена функций
В) Имена классов
Г) Имена методов и переменных экземпляров классов

Тема: Работа с библиотеками Python.

1. МОДУЛИ
1.1 Что такое модуль в Python?

- файл с расширением *.py*.

Можно условно разделить модули и программы:
- программы предназначены для непосредственного запуска
- модули для импортирования их в другие программы

Модули могут быть написаны не только на языке *Python*, но и на других языках (например *C*).

Тема: Работа с библиотеками Python.

1.2 Функции модуля

Модули выполняют как минимум три важных функции:

1. Повторное использование кода
1. Управление адресным пространством
1. Глобализация сервисов и данных

Тема: Работа с библиотеками Python.

1.3 Импорт модуля

import имя_модуля1, имя_модуля2

```
>>> import math
>>> math.factorial(5)
120
```

Импорт нескольких модулей:

```
>>> import math, datetime
>>> math.cos(math.pi/4)
0.707106781186547
>>> datetime.date(2017, 3, 21)
datetime.date(2017, 3, 21)
```

Тема: Работа с библиотеками Python.

1.3 Импорт модуля

Можно задать псевдоним для модуля:
import имя_модуля as новое_имя

```
>>> import math as m
>>> m.sin(m.pi/3)
0.866025403784438
```

Можно импортировать сразу функцию:
from имя_модуля import имя_объекта *или* ***from имя_модуля import ****

```
>>> from math import cos
>>> cos(3.14)
0.999998731727539
```

Тема: Работа с библиотеками Python.

1.4 Создание модуля

Нужно создать файл, например, my_modul.py, и наполнить его необходимыми функциями:

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

теперь в главном файле импортируем модуль (при условии что они находятся в одной директории):

```
import my_modul
print(my_modul.add(5,4))
```

Тема: Работа с библиотеками Python.

2. ПАКЕТЫ

Пакет — это директория ("каталог") с файлами модулей, имеющая имя в формате "snake_case" и содержащая, помимо прочего, специальный модуль с именем "*__init__.py*".

Именно наличие этого специального файла подсказывает интерпретатору Python, что каталог следует воспринимать именно как пакет.

Если весь код структурирован в одном рутовом каталоге, все, что нужно добавить в PYTHONPATH — это рутовый каталог.

Тема: Работа с библиотеками Python.

2.1 Простейший пакет

Пусть пакет состоит из каталога package и модуля __init__.py внутри этого каталога:

```
package/
└── __init__.py
```

Файл __init__.py пусть содержит код:

```
# file __init__.py
NAME = 'super_package'
```

Это, хотя и небольшой, но уже полноценный пакет. Его можно импортировать так же, как мы импортировали бы модуль:

```
import package

print(package.NAME)
```

Тема: Работа с библиотеками Python.

2.2 Содержимое пакета

Положим в пакет еще два модуля:

```
package/
├── constants.py
├── functions.py
└── __init__.py
```

Содержимое модуля **constants.py**:

```
# file constants.py
PERSON = 'Alice'
```

Содержимое модуля **functions.py**:

```
# file functions.py
def greet(who):
    print('Hello, ' + who + '!')
```

Тема: Работа с библиотеками Python.

2.2 Содержимое пакета

Когда пакет содержит другие модули, кроме *__init__.py*, то их можно импортировать по их именам.

```
import package.functions
import package.constants

package.functions.greet(package.constants.PERSON)  # => Hello, Alice!
```

Но писать имя пакета и имя модуля каждый раз — утомительно! Давайте импортируем саму функцию и аргумент:

```
from package.functions import greet
from package.constants import PERSON

greet(PERSON)  # => Hello, Alice!
```

Тема: Работа с библиотеками Python.

3. ВСТРОЕННЫЕ МОДУЛИ

Встроенные модули— это те модули, которые встроены непосредственно в интерпретатор.

Список встроенных модулей зависит от дистрибутива Python, а найти этот список можно в переменной:

sys.builtin_module_names

Тема: Работа с библиотеками Python.

4. КАК РАБОТАЮТ ИМПОРТЫ

При импорте модуля Python выполняет весь код в нём.

1. Интерпретатор сначала ищет встроенный модуль с таким именем
1. Далее ищет файл с именем my_module.py в текущем каталоге
1. Затем в каталогах, указанных в переменной окружения PYTHONPATH
1. Затем в зависящих от платформы путях по умолчанию, а также в специальных файлах с расширением '.pth', которые лежат в стандартных каталогах

Каталоги, в которых осуществляется поиск, можно посмотреть в переменной sys.path.

Тема: Работа с библиотеками Python.

4. КАК РАБОТАЮТ ИМПОРТЫ

Чтобы увидеть содержимое sys.path, запустите этот код:

```
import sys
print(sys.path)
```

Программист может внести изменения в PYTHONPATH и в '.pth', добавив туда свой путь:

```
sys.path.append(/home/my/lib/python)
```

Тема: Работа с библиотеками Python.

4. КАК РАБОТАЮТ ИМПОРТЫ

Все, что мы импортируем из модуля можно получить через функцию *dir():*

```
>>> dir(sys)
['__displayhook__', '__doc__', '__egginsert', '__excepthook__', '__name__',
...
'stderr', 'stdin', 'stdout', 'subversion', 'version', 'version_info']
```

Тема: Работа с библиотеками Python.

5. ФАЙЛЫ ЗАВИСИМОСТЕЙ
5.1 Зачем?

Любое приложение обычно имеет набор зависимостей, которые необходимы для работы этого приложения.

Файл требований - это способ указать и установить конкретный набор зависимостей пакета одновременно (если у вас нет virtualenv).

5.2 Как это сделать?

$ pip freeze > requirements.txt

А чтобы заказчику быстро установить все требуемые библиотеки python в новом окружении достаточно выполнить команду

pip install -r requirements.txt

Тема: Работа с библиотеками Python.

Упражнения

1. С помощью модуля math распечатайте факториал числа 7 и округлите до ближайшего целого числа число ПИ.

1. Допишите две функции умножения и деления в модуль, который мы писали ранее в качестве примера

1. Посчитайте количество функций в модуле math

1. Напишите логическую функцию after, которая принимает в качестве параметров два объекта Time, t1 и t2, и возвращает True, если t1 следует за t2 хронологически, и False, если это не так.

У класса Time должно быть три атрибута: hours, minutes, seconds
Класс Time и функция after должны быть инициализированы в отдельном файле my_time.py(), а вызваны из main.py

Тема: Работа с библиотеками Python.

Рефлексия

1. Что мы сегодня узнали?
2. Чему сегодня научились?
3. В чем отличие модулей от программ?
4. Как работают импорты?
5. В каких случаях необходимо создавать файл зависимостей?