Урок 5

Модули и библиотеки

Понятие модуля. Создание модулей. Пространство имён модулей. Дополнительные возможности модулей. Библиотеки os, sys.

Модули

Теория

Каждый файл – это отдельный модуль и модули могут импортировать другие модули для доступа к именам, которые в них определены. Обработка модулей выполняется двумя инструкциями и одной важной функцией:

import – Позволяет клиентам (импортерам) получать модуль целиком.
from – Позволяет клиентам получать определенные имена из модуля.
imp.reload – Обеспечивает возможность повторной загрузки модуля без остановки интерпретатора.
UPD. Начиная с версии Python 3.4 **imp.reload()** ***запрещен*****.** Вместо этого **– importlib.reload().**

Модули обеспечивают простой способ организации компонентов в систему автономных пакетов переменных, известных как пространства имен. Все имена, определяемые на верхнем уровне модуля, становятся атрибутами объекта импортируемого модуля.

Модули играют как минимум три роли:

1. Повторное использование программного кода.
2. Разделение системы пространств имен.
3. Реализация служб или данных для совместного использования.

Модули позволяют разделять программу на логические части (в виде отдельных файлов). И использовать объекты из одного модуля (файла) в других.

Подключение модуля

Для получения доступа к объектам другого файла (модуля), нужно воспользоваться командой import.

Например, мы хотим воспользоваться функциями модуля math:

```
import math
# sqrt() – это функция для вычисления кв.корня, которая находится в модуле math
print(math.sqrt(4))
```

Импортирование в языке Python – это не просто включение текста одного файла в другой. Это самые настоящие операции времени выполнения, которые выполняют следующие действия, когда программа впервые импортирует заданный файл:

1. Отыскивают файл модуля.
2. Компилируют в байт-код (если это необходимо).
3. Запускают программный код модуля, чтобы создать объекты, которые он определяет.

```
# Другой вариант подключения модуля
from math import sqrt
# Из модуля math мы подключили только функцию sqrt().
# Она доступна без math., напрямую
print(sqrt(4))
# Импортируем несколько функций сразу
from math import sqrt, sin, cos
print(sin(0.2))
```

Так же, как и инструкция def, инструкции import и from являются выполняемыми инструкциями, а не объявлениями времени компиляции. Они могут вкладываться в условные инструкции if,

присутствовать в объявлениях функций def и так далее, и они не имеют никакого эффекта, пока интерпретатор не достигнет их в ходе выполнения программы. Другими словами, импортируемые модули и имена в них не будут доступны, пока не будут выполнены соответствующие инструкции import или from. Подобно инструкции def, **import** и **from** – это явные ***операции присваивания***.

Все, что уже обсуждалось ранее, в равной степени применимо и к модулям. Например, имена, копируемые инструкцией from, становятся ссылками на разделяемые объекты, как и в случае с аргументами функций, повторное присваивание полученному имени не оказывает воздействия на модуль, откуда это имя было скопировано, но модификация изменяемого объекта может оказывать воздействие на объект в модуле, откуда он был импортирован.

Разделение пространства имен

В текущее пространство имён (в тот файл, в котором происходит импорт) можно импортировать весь модуль, воспользовавшись инструкцией:

```
>>>from math import *
```

Чаще всего, это плохой подход, т.к. имена функций модуля, могут конфликтовать с именами вашего или других подключенных (импортированных) модулей.

```
print(cos(0.4))
# не совсем понятно, из какого модуля данная функция,
# особенно если подключено много модулей
```

Удобство import math – заключается в том, что вы создаёте пространство имен math, и избегаете конфликтов (при одинаковых именах функций и классов в различных модулях).
Например, в module1 есть функция do_this() и в module2 есть функция do_this:

```
>>> import module1
>>> import module2
```

Используем одноименные, но разные функции из разных модулей:

```
>>> module1.do_this()
>>> module2.do_this()
```

Поиск модуля

Прежде всего, интерпретатор должен определить местонахождение файла модуля, указанного в инструкции import. Обратите внимание, что имена файлов в инструкции import в примерах из предыдущих разделов указаны без расширения .py и без пути к каталогу: вместо записи в виде, например, import c:\dir1\b.py, инструкция записывается просто – import b. Фактически допускается указывать лишь простые имена – путь к каталогу и расширение файла должны быть опущены, потому что для поиска файла, соответствующего имени, указанному в инструкции import, интерпретатор использует стандартный путь поиска модулей.

В общих чертах пути поиска модулей в языке Python выбираются из объединенных данных следующих основных источников. Некоторые из них предопределены, а некоторые можно настроить и тем самым сообщить интерпретатору, где выполнять поиск:

1. Домашний каталог программы.
2. Содержимое переменной окружения PYTHONPATH (если таковая определена).

3. Каталоги стандартной библиотеки.
4. Содержимое любых файлов с расширением .pht (если таковые имеются).

В конечном итоге объединение этих четырех компонентов составляет sys.path – список строк с именами каталогов.

Эти пути можно легко посмотреть:

```
import sys
print('sys.path = ', sys.path)
```

Выполните этот код у себя, чтобы посмотреть свои пути поиска модулей.

Поиск происходит поочерёдно, если ни в одной из директорий sys.path не будет найден файл <имя модуля>.py, то получите ошибку импорта.

Создание собственных модулей

Т.к. модули – это просто python-файлы, можно легко создавать собственные библиотеки функций.

Давайте создадим простой модуль с двумя функциями и импортируем его.

1. В корневой директории создайте папку /libs.
2. Внутри папки /libs разместите файл my_fisrt_lib.py.
3. В этом файле опишем пару тестовых функций.

```
# Демонстрационный модуль
def do_something():
    return "I'm func do_something in module lib1"
def more_then_one(num):
    return num > 1
```

4. В корне проекта создадим файл run.py

```
import libs.my_fisrt_lib as my_lib
print(my_lib.do_something())
print(my_lib.more_then_one(6))
```

Инструкцию as используем, чтобы обращаться к модулю по более удобному имени.

Обратите внимание! Раз модуль находится в папке libs, то нужно указать **libs.** при импорте.

Импорт каждого модуля выполняется только ОДИН раз! Это сделано для экономии ресурсов.

Как видите, все очень просто!

Дополнительные возможности выбора модулей

Обычно операция импорта работает именно так, как описывается в данном разделе, – она отыскивает и загружает файлы, находящиеся на вашей машине. Однако вполне возможно переопределить большую часть того, что делает операция импорта, используя то, что называется программными ловушками импорта. Эти ловушки могут использоваться, чтобы придать операции импорта дополнительные полезные возможности, такие как загрузка файлов из архивов, расшифровывание и так далее.

Фактически, сам интерпретатор Python использует эти ловушки, чтобы обеспечить возможность извлечения импортируемых компонентов из ZIP-архивов, заархивированные файлы автоматически извлекаются во время импорта, когда в пути поиска выбирается файл с расширением .zip. Например, один из каталогов стандартной библиотеки в списке sys.path, представленном выше, на сегодняшний день является файлом .zip. За дополнительной информацией обращайтесь к описанию

встроенной функции import в руководстве по стандартной библиотеке Python – настраиваемому инструменту, которым в действительности пользуется инструкция import.

Подробнее в этой статье. Но пока не рекомендую серьёзно обращать на это внимание, особенно если вы пока не планируете заниматься разработкой собственных профессиональных модулей.

Модуль OS

Модуль os предоставляет множество функций для работы с операционной системой, причём их поведение, как правило, не зависит от ОС, поэтому программы остаются переносимыми.

С помощью функций модуля os можно:

- Узнать тип ОС;

```
import os
print('os.name = ', os.name)
```

Для Windows будет выведено: nt.

- Получить полный путь до текущего файла (файла, из которого производится запуск);

```
print('os.getcwd() = ', os.getcwd())
```

- Создать новую директорию.

```
dir_path = os.path.join(os.getcwd(), 'NewDir')
try:
    os.mkdir(dir_path)
except FileExistsError:
    print('Такая директория уже существует')
```

И многое другое, обо всех функция модуля os читайте здесь.

Модуль SYS

Модуль sys обеспечивает доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором python.

```
import sys
# Список аргументов запуска скрипта,
# первым аргументом является полный путь к файлу скрипта
print('sys.argv = ', sys.argv)
# Список путей для поиска модулей
print('sys.path = ', sys.path)
# Полный путь к интерпретатору
print('sys.executable = ', sys.executable)
# Словарь имён загруженных модулей
print('sys.modules = ', sys.modules)
# Информация об операционной системе
print('sys.platform = ', sys.platform)
while True:
    key = input("press 'q' to Exit")
    if key == 'q':
        sys.exit()
        # Вызов данной функции мгновенно завершает работу модуля (скрипта)
```

Запуск скрипта с параметрами

Любой python-скрипт можно запустить с дополнительными параметрами.

```
    > python3 my_script.py param1 param2 param3
```

Эти параметры легко можно извлечь в теле скрипта, используя модуль sys.

```
import sys
print('sys.argv = ', sys.argv)
```

Получим:

```
sys.argv = ['with_args.py', 'param1', 'param2', 'param3']
```

Это позволяет создавать более гибкие скрипты для работы в командной строке.

Напишем небольшой демо-скрипт с использованием дополнительных параметров запуска.

```
import os
import sys
print('sys.argv = ', sys.argv)

def print_help():
    print("help - получение справки")
    print("mkdir <dir_name> - создание директории")
    print("ping - тестовый ключ")

def make_dir():
    if not dir_name:
        print("Необходимо указать имя директории вторым параметром")
        return
    dir_path = os.path.join(os.getcwd(), dir_name)
    try:
        os.mkdir(dir_path)
        print('директория {} создана'.format(dir_name))
    except FileExistsError:
        print('директория {} уже существует'.format(dir_name))
def ping():
    print("pong")

do = {
    "help": print_help,
    "mkdir": make_dir,
    "ping": ping
}
try:
    dir_name = sys.argv[2]
except IndexError:
    dir_name = None
try:
    key = sys.argv[1]
except IndexError:
    key = None
if key:
    if do.get(key):
        do[key]()
    else:
        print("Задан неверный ключ")
        print("Укажите ключ help для получения справки")
```

Поэкспериментируйте с данным скриптом, добавьте несколько собственных параметров запуска.

Обратите внимание на использование словаря do для запуска нужной функции и на обработку исключений.

Домашнее задание

1. Смотреть здесь https://github.com/GeekBrainsTutorial/Python_lessons_basic/tree/master/lesson05

Большинство заданий делятся на три категории easy, normal и hard.

- easy - простенькие задачи, на понимание основ.
- normal - если вы делаете эти задачи, то вы хорошо усвоили урок.
- hard - наиболее хитрые задачи, часто с подвохами, для продвинутых слушателей.

Если вы не можете сделать normal задачи - это повод пересмотреть урок, перечитать методичку и обратиться к преподавателю за помощью.

Если не можете сделать hard - не переживайте, со временем научитесь.

Решение большинства задач будем разбирать в начале каждого вебинара.

Дополнительные материалы

Всё то, о чем сказано здесь, но подробнее:

1. Глубоко в import
2. Модуль os

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Учим python качественно(habr)
2. Самоучитель по python
3. Книга Лутц М. “Изучаем Python” (4-е издание).