Лекція 19

Модулі і пакети (продовження)

Завантаження from з *

Формат інструкції:

```
from <Назва модуля> import *
```

from дозволяє імпортувати з модуля всі ідентифікатори. Для прикладу імпортуємо всі ідентифікатори з модуля math.

Приклад 12. Імпорт усіх ідентифікаторів з модуля

```
# Імпортуємо всі ідентифікатори з модуля math

from math import *
print (pi) # Вивід числа pi
print(floor(5.49)) # Викликаємо функцію floor ()
Результат виконання:
3.141592653589793
5
```

Недоліки повного імпортування

Необхідно враховувати, що імпортування всіх ідентифікаторів з модуля може порушити простір імен головної програми.

Причина

1.У різних модулях можуть міститися ідентифікатори з однаковими назвами.

2. Ідентифікатори, що мають однакові імена з головною програмою, будуть перезаписані.

Приклад перезапису ідентифікаторів

Створимо два модулі й підключимо їх за допомогою інструкцій `from` і `import`.

Приклад 13.

```
# Вміст файлу module1.py
s = "Значення з модуля module1"
# Вміст файлу module2.py
s = "Значення з модуля module2"
```

```
# початковий код основної програми main.py
from module1 import *
from module2 import *
import module1, module2
print(s)  # Виведе: "Значення з модуля module2"
print(module1.s) # Виведе: "Значення з модуля module1"
print(module2.s) # Виведе: "Значення з модуля module2"
```

Приклад в файлі main.py

Шляхи пошуку модулів

Дотепер ми розміщали модулі в одній папці з файлом основної програми. У цьому випадку немає необхідності набудовувати шляхи пошуку модулів, тому що папка з файлом, що виконується, автоматично додається в початок списку шляхів.

Список `sys.path` містить шляхи пошуку, одержувані з наступних джерел:

- шлях до поточного каталогу з кодом основної програми;
- значення змінної оточення PYTHONPATH.

Одержати повний список шляхів пошуку дозволяє наступний код:

Приклад 14.

```
>>> import sys # Підключаємо модуль sys
>>> sys.path # path містить список шляхів
пошуку модулів
['C:\\Program Files
(x86)\\Jetbrains\\Pycharm Edu
2.0.4\\helpers\\pydev', 'C:\\Program Files
(x86)\\Jetbrains\\Pycharm Edu
2.0.4\\helpers\\pydev', 'C:\\Program Files
(x86)\\Python35-32\\python35.zip',
'C:\\Program Files (x86)\\Python35-
32\\Dlls', 'C:\\Program Files
(x86)\\Python35-32\\lib', 'C:\\Program Files
(x86)\\Python35-32', 'C:\\Program Files
(x86)\\Python35-32\\lib\\site-packages',
'C:\\LECTURCE18']
```

Модифікація змінної PYTHONPATH

1.Для додавання змінної в меню **Пуск** вибираємо пункт **Панель керування** (або Настроювання І Панель керування).
2.У вікні, що відкрилося, вибираємо пункт Система.
3.Клацаємо на посиланні Додаткові параметри системи.
4.Переходимо на вкладку Додатково й натискаємо кнопку Змінні середовища.
5.У розділі Змінні середовища користувача натискаємо кнопку Створити.
6.У поле Ім'я змінної вводимо PYTHONPATH, а в полі Значення змінної задаємо шляхом до папки – наприклад, `C:\pypackages`.

Приклад в файлі example2.py

Файли з розширенням *.pth

Для задавання нових шляхів до файлів необхідно створити файл з розширенням *.pth і розмістити його в каталозі:

C:\Program Files(x86)\Python35-32\Lib\site-packages.

1.Назви таких файлів можуть бути довільними, головне, щоб вони мали розширення pth.

2.Кожний шлях повинен бути розташований на окремому рядку.

3.Назву диску писати з великої літери

Для прикладу створіть файл `mypath.pth` у каталозі

Приклад 15.
`C:\Program Files(x86)\Python35-32\Lib\site-packages\mypath.pth`

з наступним вмістом:

```
# Це мої нові  шляхи на файли Python
C:\pyfiles
C:\pyprogs
```

Приклад в файлі distance.py

Правила роботи з файлом mypath.pth

1. Каталоги повинні існувати, а якщо ні, то вони не будуть додані в список `sys.path`.
2. При пошуку модуля список `sys.path`, проглядається зліва направо.
3. Пошук припиняється після першого знайденого модуля. Таким чином, якщо в каталогах `C:\pyfiles` і `C:\pypackages` існують однойменні модулі, то буде використовуватися модуль із папки, яка стоїть перед даною папкою.
4. Список `sys.path` можна змінювати із програми за допомогою списковых методів.

Наприклад, додати каталог у кінець списку можна за допомогою методу `append()`, а в його початок – за допомогою методу `insert()`.

Приклад 15. **Зміна списку шляхів пошуку модулів**

```
import sys
>>> sys.path.append(r"C:\pyLecture19")
# Додаємо в кінець списку
>>>sys.path.insert(0, r"pyLecture1")
# Додаємо в початок списку
>>> print(sys.path)
```

Результати виконання:

```
['C:\\pyLecture1', 'C:\\PYTHON',…,
'C:\\pyfiles', 'C:\\pypackages',
'C:\\pyLecture19'
```

У цьому прикладі ми додали папку `'C:\\pylecture1'` у початок списку.

```
sys.path.append("'C:\\pylecture1")
```

Приклад в файлі dynamic.py

Повторне завантаження модулів

Як ви вже знаєте, модуль завантажується тільки один раз при першій операції імпорту.

Усі наступні операції імпортування цього модуля будуть повертати вже завантажений об'єкт модуля, навіть якщо сам модуль був змінений.

Щоб повторно завантажити модуль, слід скористатися функцією **`reload()`** з модуля **`importlib`**.

Формат функції:

```
from importlib import reload

reload(<Об'єкт модуля>)
```

~~**im**~~

Приклад 16. Розглянемо файли pyrel.py та testrel.py

Файл pyrel.py є головним файлом програми, а файл testrel.py буде імпортовано в pyrel.py як модуль.

pyrel.py:

```
import testrel, importlib
print(testrel.x)
```

testrel.py:

```
x=200
#x='File testrel reloaded'
```

Якщо під час роботи програми змінювати вміст файлу testrel.py, то це не ніяк не позначиться на роботі програми.

Буде працювати старий файл testrel.py

Зміна модуля «на льоту» потребує перезавантаження модуля

Файл pyrel.py, який містить інструкцію перезавантаження:

pyrel.py:

```
import testrel, importlib
importlib.reload(testrel)
print(testrel.x)
```

Змінений вміст файлу testrel.py:

testrel.py:

```
#x=200
x='File testrel reloaded'
```

Приклад в файлах pyrel.py та testrel.py

Особливості застосування функції `reload`

1.При використанні функції `reload()` слід враховувати, що ідентифікатори, імпортовані за допомогою інструкції **`from`**, перезавантажені не будуть.

2.Крім того, повторно не завантажуються скомпільовані модулі, написані на інших мовах програмування, – наприклад, мовою C.

ПАКЕТИ

Означення пакета

Пакетом називають каталог з модулями, у якому розташований файл ініціалізації __init__.py.

Файл ініціалізації може бути порожнім або містити код, який буде виконаний при першому доступі до пакета.

У будь-якому разі він обов'язково повинен бути присутнім всередині каталогу з модулями.

Приклад структури файлів і каталогів:

Приклад 1

```
first.py # Основний файл з програмою
mypack\# Папка на одному рівні вкладеності з first.py
    __init__.py # Файл ініціалізації
    sample1.py # Модуль mypack\sample1.py
    subpack\ # Вкладена папка
        __init__.py # Файл ініціалізації
        subsamp1.py
        subsamp2.py
```

Вміст файлу **_init_.py** для пакету **mypack** наведений в наступному прикладі.

Приклад 2. **Вміст файлу __init__.py**

```
print("__init__",__name__)
```

```
Результат: __init__ __mypack__
```

Заповнимо модулі sample1.py, subsamp1.py і subsamp2.py як показано в прикладі 3

Приклад 3. Вміст модулів sample1.py, subsamp1.py і subsamp2.py

```
msg = "Модуль {0}".format(__name__)
```

Імпорт модулів в основний файл

Приклад 4.

```
print("Доступ до модуля sample1")
import mypack.sample1 as s1
print(s1.msg)
print(s1.__name__)

print("\nДоступ до модуля subsump1")
import mypack.subpack.subsamp1 as ss1
print(ss1.msg)
print(ss1.__name__)

print("\nДоступ до модуля subsump2")
import mypack.subpack.subsamp2 as ss2
print(ss2.msg)
print(ss2.__name__)
```

Приклад 5. Результати роботи програми first

```
Доступ до модуля sample1
__init__ mypack
Модуль mypack.sample1
mypack.sample1

Доступ до модуля subsamp1
Модуль mypack.subpack.subsamp1
mypack.subpack.subsamp1

Доступ до модуля subsamp2
Модуль mypack.subpack.subsamp2
mypack.subpack.subsamp2
```

Приклад в файлі first.py та каталозі mypack

Порядок доступу до модулів пакета

Як видно з прикладу, пакети дозволяють розподілити модулі по каталогах.

Щоб імпортувати модуль, розташований у вкладеному каталозі, необхідно вказати шлях до нього, перелічивши імена каталогів через крапку.

Якщо модуль розташований у каталозі C:\LECTURE19\mypack\subpack\, то шлях до нього повинен бути записаний так: **`mypack.subpack`** за умови, що C:\LECTURE19 – це папка проекту.

Застосування інструкції import

При використанні інструкції import шлях до модуля повинен включати не тільки назви каталогів, але й назву модуля без розширення:

```
import mypack.subpack.subsamp1
```

Одержати доступ до ідентифікаторів всередині імпортованого модуля можна в такий спосіб:

Приклад 6.

```
import mypack.subpack.subsamp1
print(mypack.subpack.subsamp1.msg)
```

Результат роботи:

```
Модуль mypack.subpack.subsamp1
```

Псевдоніми

При використанні довгих ідентифікаторів виникають незручності.

Можна створити псевдонім, указавши його після ключового слова as, і звертатися до ідентифікаторів модуля через нього:

Приклад 7.

```
import mypack.subpack.subsamp1 as ss1
print(ss1.msg)
```

Результат роботи:

```
Модуль mypack.subpack.subsamp1
```

Застосування інструкції from

При використанні інструкції from можна імпортувати як об'єкт модуля, так і визначені ідентифікатори з модуля.

Щоб імпортувати об'єкт модуля, його назву слід указати після ключового слова import:

Приклад 8

```
from mypack.subpack import subsamp1
print(subsamp1.msg)
```

Результат роботи:

```
Модуль mypack.subpack.subsamp1
```

Імпортування визначених ідентифікаторів

Для імпортування тільки визначених ідентифікаторів назва модуля вказується в складі шляху, а після ключового слова import через кому перелічуються ідентифікатори.

Додамо в модуль subsamp1 :

```
msg1 = "Це msg1 з модуля {0}".format(__name__)
msg2 = "Це msg2 з модуля {0}".format(__name__)
```

Приклад 9

```
from mypack.subpack.subsamp1 import msg, msg1,msg2
print (msg)
print (msg1)
print (msg2)
```

Результат роботи: Приклад у файлі second.py

```
Модуль mypack.subpack.subsamp1
Це msg1 з модуля mypack.subpack.subsamp1
Це msg2 з модуля mypack.subpack.subsamp1
```

Імпортування всіх ідентифікаторів

Якщо необхідно імпортувати всі ідентифікатори з модуля, то після ключового слова import вказують символ *:

Приклад 10

```
from mypack.subpack.subsamp1 import *
print (msg)
print (msg1)
print (msg2)
```

Результат роботи:

Модуль `mypack.subpack.subsamp1`

Це msg1 з модуля `mypack.subpack.subsamp1`

Це msg2 з модуля `mypack.subpack.subsamp1`

Імпортування кількох модулів одночасно

Інструкція from дозволяє також імпортувати відразу кілька модулів з пакета.

Для цього всередині файлу ініціалізації __init__.py в атрибуті __all__ необхідно вказати список модулів, які будуть імпортуватися за допомогою виразу:

from *пакет* **import** *

Змінимо вміст файлу mypack.subpack1._init_.py :

```
__all__ = ["subsamp1", " subsamp2"]
```

Приклад в модулі third.py

Приклад 11.

```
from mypack.subpack1 import *
print(subsamp1.msg)
print(subsamp1.msg1)
print(subsamp1.msg2)
print(subsamp2.msg)
```

Результат роботи:

```
Модуль mypack.subpack1.example1
Це msg1 з модуля mypack.subpack1.subsamp1
Це msg2 з модуля mypack.subpack11.subsamp1
Модуль mypack.subpack1.subsamp2
```

Після ключового слова **from** вказується лише шлях до каталогу без імені модуля. У результаті виконання інструкції from усі модулі, зазначені в списку __all__, будуть імпортовані в простір імен файлу **third.py**.

Імпортування модулів всередині пакета

Інструкція **from** підтримує відносний імпорт модулів. Щоб імпортувати модуль, розташований у тому ж каталозі, перед назвою модуля вказують крапку:

from .module **import** *

Щоб імпортувати модуль, розташований у батьківському каталозі, перед назвою модуля вказують дві крапки: **from** ..module **import** *

Якщо необхідно звернутися ще рівнем вище, те вказують три крапки: **from** ... module **import** *

Чим вище рівень, тим більше крапок необхідно вказати. Після ключового слова from можна вказувати одні тільки крапки – у цьому випадку ім'я модуля вводиться після ключового слова import.

from . . **import** module

Приклади відносного імпорту модулів

Розглянемо відносний імпорт на прикладі. Для цього створимо модуль relative.py, як показано в прикладі

Приклад 12. Вміст модуля relative.py

```
# Імпорт модуля subsamp1 з поточного
каталогу
from .import subsamp1 as ss1
var1 ="Значення з: {0}".format(ss1.msg)
from  .subsamp1 import  msg as ms1
var2 = "Значення з: {0}".format(ms1)
#Імпорт модуля sample1 з батьківського
каталогу
from ..import sample1 as smp
var3 = "Значення з: {0}".format(smp.msg)
from  ..sample1 import msg as m
var4 = "Значення з: {0}".format(m)
```

Тепер розглянемо вміст основного файлу four.py і запустимо його.

Приклад 13. Вміст файлу main.py

```
from mypack.subpack import relative as ss2
print(ss2.var1)
print(ss2.var2)
print(ss2.var3)
print(ss2.var4)
```

Результат роботи:

```
Значення з: Модуль mypack.subpack.subsamp1
Значення з: Модуль mypack.subpack.subsamp1
Значення з: Модуль mypack.sample1
Значення з: Модуль mypack.sample1
```

Пакет docutils

Цей пакет і набір утиліт поки що не входить у стандартну поставку Python, однак про нього потрібно знати тим, хто прагне швидко готовити документацію (посібники користувача) для своїх модулів.

Цей пакет використовує спеціальну мову розмітки (Restructuredtext), з якого потім легко виходить документація у вигляді HTML, Latex і в інших форматах. Текст у форматі RST легко читати й у початковому виді. Із цим інструментом можна познайомитися на http://docutils.sourceforge.net

Пакет distutils

Даний пакет надає стандартний шлях для поширення власних Python-Пакетів. Досить написати невеликий конфігураційний файл setup.py, що використовує distutils, і файл із перерахуванням файлів проекту MANIFEST.in, щоб користувачі пакета змогли його встановити командою

```
python setup.py install
```

Тонкощі роботи з distutils можна вивчити по документації.