Akademia Pythona

V Moduły

KN Pythona - Kurs Pythona

KN Pythona wita na kursie Pythona.

Plan:

- Moduły: wprowadzenie
- Podstawy tworzenia modułów
- Pakiety modułów
- Zaawansowane zaganienia związane z modułami

Moduły

Rola modułów:

- Ponowne wykorzystanie kodu
- Dzielenie przestrzeni nazw systemu
- Implementowanie współdzielonych usług oraz danych

Struktura programu

Program w Pythonie najczęściej składa się z plików tekstowych zawierających instrukcje Pythona. Program ustrukturyzowany jest jako jeden główny plik najwyższego poziomu wraz z plikami dodatkowymi (modułami).

Moduły - wprowadzenie

b.py

```
def say(text):
    print(f'Saying: {text}')
```

Moduły - wprowadzenie

a.py

```
import b

b.say('hello!')
```

Jak działa importowanie?

Importowanie:

- Odnalezienie pliku modułu
- Skompilowanie go do kodu bajtowego
- Wykonanie kodu modułu w celu utworzenia definiowanych przez niego obiektów

Odnajdywanie pliku

```
import C:\katalog1\b.py # niepoprawnie

import b # poprawnie
```

Odnajdywanie pliku

Ścieżka wyszukiwania modułów:

- Katalog główny programu
- Katalogi PYTHONPATH
- Katalogi biblioteki standardowej
- Zawartość wszystkich plików .pth

Kompilowanie

Python sprawdza daty plików źródłowych i skompilowanych, a następnie dokonuje kompilacji jeżeli jest taka potrzeba. Pliki skompilowane zapisywane są z rozszerzeniem .pyc

Wykonanie

W trakcie importowania Python wykona wszystkie instrukcje z modułu od góry do dołu, a nastepnie przypisze zakres globalny modułu do zmiennej o nazwie modułu.

Moduły

```
import sys
sys.path
```

Import b

Wybór pliku modułu dla **import b**:

- Plik z kodem źródłowym o nazwie b.py
- Plik z kodem bajtowym o nazwie b.pyc
- Katalog o nazwie b (importowanie pakietów)
- Skompilowany moduł rozszerzenia np. C, C++ (b.so, b.dll, b.pyd)
- Skompilowany moduł wbudowany napisany w języku C i statycznie dołączony do Pythona
- Komponent pliku ZIP rozpakowywany automatycznie po zaimportowaniu
- Obraz z pamięci (zamrożone pliki wykonywalne)
- Klasę języka JAVA (Jython)
- Komponent .NET (IronPython)

Nazewnictwo modułów

Nazwy modułów muszą być poprawnymi nazwami zmiennych Pythona, jeżeli chcemy je importować.

Tworzenie modułów

module1:

```
def printer(text):
    print(text)
```

Tworzenie modułów

```
import module1
module1.printer('Hello!')
```

Tworzenie modułów

```
from module1 import printer
printer('Hello!')
```

Tworzenie modułów

```
from module1 import *
printer('Hello!')
```

Moduły

Operacja importowania odbywa się tylko raz.

Moduły

Instrukcje import oraz from:

- Instrukcja import przypisuje cały obiekt modułu do jednej nazwy.
- Instrukcja from przypisuje jedną lub więcej zmiennych do obiektów o tych samych nazwach w innym module.

Przestrzenia nazw modułów

Pliki generują przestrzenie nazw:

- Instrukcje modułów wykonywane są przy pierwszej operacji importowania.
- Przypisania na najwyższym poziomie pliku trworzą atrybuty modułów.
- Dostęp do przestrzeni nazw modułu odbywa się za pomocą atrybutu **dict** lub dir(M)

Moduły

module2.py

```
print('Started loading...')
import sys
name = 42
def func(): pass
class klass: pass
print('Finished loading...')
```

Moduły

```
import module2

module2.sys
module2.name
module2.func
module2.klass
list(module2.__dict__.keys())
```

Podstawy przeładowywania modułów

```
import module

from importlib import reload

reload(module)
```

Podstawy przeładowywania modułów

Przeładowywanie:

- Funkcja reload wykonuje nowy kod pliku modułu w bieżącej przestrzeni nazw modułu.
- Przypisania najwyższego poziomu w pliku zastępują zmienne z nowymi wartościami.
- Przeładowanie ma wpływ na każdy kod wykorzystujący instrukcję import do pobrania modułów.
- Przeładowanie ma wpływ jedynie na przyszły kod wykorzystujący instrukcję **from**.

Podstawy importowania pakietów

```
import dir1.dir2.module

from dir1.dir2 import module

# zakładając dir0/dir1/dir2/module.py
# dir0 w ścieżce wyszukiwania Pythona
```

Katalogi pakietów

Katalogi pakietów muszą zawierać plik **init**.py

Katalogi pakietów

```
dir0\
    dir1\
        __init__.py
        dir2\
            __init__.py
            module.py
```

Pliki **init**.py

Rola plików **init**.py:

- Inicjalizacja pakietów
- Inicjalizacja przestrzeni nazw modułu
- Definicja listy **all**

Importy względne

```
import string
from mypkg import string
from . import string
from .. import string
```

Minimalizacja niebezpieczeństw

```
_non_exported = 12
exported = 13

__all__ = ['this', 'will', 'be', 'exported']
```

Mieszany tryb użycia

```
...


if __name__ == '__main__':
    main()
```

Rozszerzenie as dla instrukcji import oraz from

```
import module
name = module
func2 = module.func1
del module
```

Rozszerzenie as dla instrukcji import oraz from

```
import module as name
from module import func1 as func2
```

Metaprogramy

```
import M
M.name
M.__dict__['name']
sys.modules['M'].name
getattr(M, 'name')
```

Importowanie modułów

```
import 'module' # Błąd

from importlib import import_module
import_module('module')

exec('import module')
```

Projektowanie modułów

Reguły projektowania modułów:

- W Pythonie zawsze jesteśmy w module.
- Należy minimalizować połączenia pomiędzy modułami w postaci zmiennych globalnych.
- Maksymalizacja spójności modułów: jeden cel.
- Moduły powinny rzadko modyfikować zmienne z innych modułów.