

Modularyzacja programu w języku Python - moduły

Zagadnienia poruszane w ramach modułu

- Informacje podstawowe o modułach w języku Python
- Prawidłowy podział programu na moduły
- Przestrzenie nazw w modułach
- Ładowanie modułów na żądanie
- Pakiety modułów
- Implikacje i niespodzianki w stosowaniu modułów

Informacje podstawowe o modułach w języku Python

- Modułów w języku Python używa się aby osiągnąć:
 - Ponowne użycie kodu w nowych aplikacjach
 - Podział przestrzeni nazw na oddzielne bloki
 - Wprowadzenie współdzielenia usług i danych pomiędzy aplikacjami i częściami aplikacji
- Moduły w języku Python mogą być tworzone jako:
 - Programy w języku Python
 - Rozszerzenia w postaci bibliotek ładowanych przez interpreter (*.so i *.dll)
- Moduły wyszukiwane są w ścieżce umieszczonej w zmiennej PYTHONPATH
- Nazwy modułów powinny spełniać reguły nazw zmiennych

Sterowanie ładowaniem i użytkowaniem modułu

- Do obsługi modułów służą w języku Python instrukcje:
 - **import** – ładowanie modułu
 - **from** – podanie części ładowanych
 - **as** – zmiana nazwy pobranej z modułu na inną
 - **reload** – przeładowanie modułu

Tworzenie modułu

- Po stworzeniu modułu, możemy użyć jego metod:

```

1 # plik modul.py znajdujący się w
2 # ścieżce w zmiennej PYTHONPATH
3
4 def egoista(x = 'Python'):
5     """metoda egoistycznie drukująca.."""
6     print "Ja", x

```

Składnia i wywołanie	Znaczenie
import modul modul.egoista()	Ładuje wszystkie metody, tworzy przestrzeń nazw
from modul import egoista egoista()	Ładuje jedną metodę, nie tworzy przestrzeni nazw
from modul import * egoista()	Ładuje wszystkie metody, nie tworzy przestrzeni nazw
from modul import egoista as jaja jaja()	Ładuje jedną metodę, nie tworzy przestrzeni nazw, zmienia nazwę metody na podaną

Przestrzenie nazw w modułach

- Instrukcje modułu są uruchamiane podczas pierwszego importu
- Przypisania i definicji najwyższego poziomu w module, tworzą jego atrybuty
- Przestrzeń nazw modułu można sprawdzić z użyciem funkcji **dir()** lub przeglądając atrybut słownika **__dict__**
- Odwołanie się do metody lub atrybutu możliwe jest poprzez kwalifikator oznaczony kropką: X.Y()
 - X – moduł
 - Y – funkcja lub metoda

Ładowanie modułu na żądanie

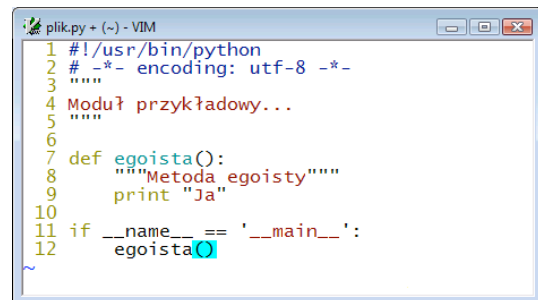
- Kod modułu wykonywany jest tylko raz w trakcie importu
- Jeśli kod modułu zmieniono lub niezbędne jest jego ponowne uruchomienie, wywołujemy instrukcję **reload(moduł)**
- Instrukcja **reload()** powoduje:
 - Ponowne uruchomienie kodu modułu w bieżącej przestrzeni nazw
 - Przypisania w module zastępują bieżące przypisania
 - Ładowanie wpływa na wszystkie części programu
 - Ponowne ładowanie ma wpływ na nowe wywołania **from**
- Wyładowanie modułu z pamięci można wykonać z pomocą **del()**

Prawidłowy podział programu na moduły

- Moduły powinny zawierać wywołania powiązane funkcjonalnie, metody służą jednemu ogólnemu celowi
- Kod modułu może być samodzielnym kodem zawierającym procedury testowe lub demonstracyjne
- Należy przestrzegać zasad dokumentowania modułów (ciągi dokumentujące)
- Należy dokładnie przemyśleć czy kod uruchamiany przy ładowaniu modułu jest absolutnie niezbędny
- Program główny działa w obrębie modułu o nazwie **'__main__'**
- Moduł powinien mieć jak najmniejsze sprzężenia z innymi

Uniwersalny kod modułu

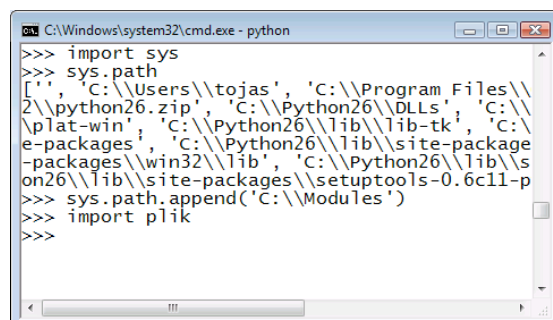
- W trakcie uruchamiania modułu, deklarowana jest zmienna `__name__` która w przypadku uruchamiania samodzielnego, ma wartość `'__main__'`
- Jeśli moduł jest ładowany do aplikacji, zmienna ta wskazuje na `'nazwa_modułu'`
- Mechanizm ten najczęściej jest stosowany do kodu testującego działanie modułu



```
1 #!/usr/bin/python
2 #-*- encoding: utf-8 -*-
3
4 Moduł przykładowy...
5
6
7 def egoista():
8     """Metoda egoisty"""
9     print "Ja"
10
11 if __name__ == '__main__':
12     egoista()
```

Ścieżka przeszukiwania modułów

- W trakcie działania aplikacji, można zmienić ścieżkę przeszukiwania modułów



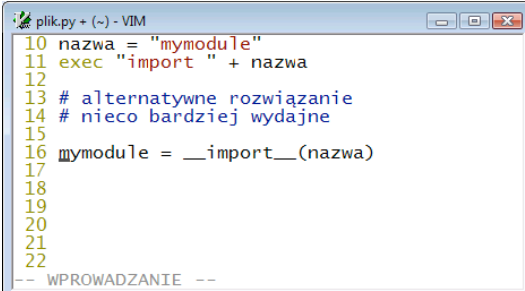
```
C:\Windows\system32\cmd.exe - python
>>> import sys
>>> sys.path
['', 'C:\\Users\\tojas', 'C:\\Program Files\\
2\\python26.zip', 'C:\\Python26\\DLLs', 'C:\\
\\plat-win', 'C:\\Python26\\lib\\lib-tk', 'C:\\
e-packages', 'C:\\Python26\\lib\\site-package
-packages\\win32\\lib', 'C:\\Python26\\lib\\s
on26\\lib\\site-packages\\setuptools-0.6c11-p
>>> sys.path.append('C:\\Modules')
>>> import plik
>>>
```

Pakiety modułów

- Większe i bardziej złożone moduły, można zapisać w oddzielnych katalogach
- Nazwa katalogu staje się wtedy nazwą modułu
- W katalogu poszukiwany jest plik `__init__.py`, który może inicjować ładowanie modułów zależnych
- W pliku `__init__.py`, można zdefiniować listę `__all__` która zawierać będzie tylko te metody i atrybuty które mają być ładowane poprzez **`from X import *`**
- Można także modyfikować zmienną `__path__` w ramach pliku `__init__.py`, aby dodać ścieżki domyślnie przeglądane w trakcie wyszukiwania modułów

Implikacje i niespodzianki w stosowaniu modułów

- Ładowanie modułu na podstawie jego nazwy, możliwe jest z użyciem metody **`exec`**
- Rekurencyjne importy mogą nie działać poprawnie. **Unikaj tego!**
- Funkcja **`reload()`** nie działa przechodnio



```
plik.py + (~) - VIM
10 nazwa = "mymodule"
11 exec "import " + nazwa
12
13 # alternatywne rozwiązanie
14 # nieco bardziej wydajne
15
16 mymodule = __import__(nazwa)
17
18
19
20
21
22
-- WPROWADZANIE --
```

Quiz

- Jaką instrukcją załadujemy moduł z zachowaniem jego przestrzeni nazw?
- Jak nazywa się zmienna którą można testować aby uczynić kod modułu możliwy do uruchamiania jako samodzielny?
- Jak nazywa się zmienna która przechowuje ścieżkę dostępu do modułów w języku Python?
- Jak działa dyrektywa **as** przy imporcie modułu?
- Do jakich celów służy wywołanie **reload()**?

Ćwiczenie

- Wykonaj ćwiczenie 5.

Podsumowanie

Informacje podstawowe o modułach w języku Python

- Moduł jest wydzielonym fragmentem programu w języku Python, który może być ładowany na żądanie
- Metody i klasy w module służą jednej grupie funkcjonalności aplikacji
- Nazwa modułu powinna spełniać warunki nazwy zmiennej języka Python

Prawidłowy podział programu na moduły

- Moduły powinny być konstruowane z jak najmniejszym powiązaniem z kodem aplikacji
- Interfejsy modułu powinny być ściśle określone
- Zadbaj o dobrą dokumentację modułu

Przestrzenie nazw w modułach

- Ładowanie modułu z użyciem **from**, nie tworzy nowej przestrzeni nazw
- W trakcie działania kodu modułu, definiowana jest zmienna **__name__** która w szczególnym przypadku uruchamiania modułu samodzielnie, przyjmuje wartość **'__main__'**

Ładowanie modułów na żądanie

- Moduł można załadować z użyciem instrukcji **import**, **from**, **as**
- Przeładowanie modułu możliwe jest z użyciem **reload()**

Pakiety modułów

- W pliku **__init__.py** można inicjować ładowanie pakietu modułów
- Można modyfikować tablice **__all__** oraz **__path__**

Implikacje i niespodzianki w stosowaniu modułów

- Rekursywne ładowanie modułów jest ryzykowne
- Tworzenie nazwy modułu do załadowania w trakcie działania aplikacji wymaga stosowania instrukcji **exec**