zaj1-blok4

October 22, 2015

1 Moduły, paczki

Plik .py nazywany jest modułem pythona.

Powiedzmy że napisałem dwie bardzo przydatne funkcje i chcę je jakoś razem umieścić tak by inni mogli z nich korzystać. By je spakować po prostu zapisuję te dwie funkcje do pliku.

In [1]:
```
def _fib_generator():
    """
    Ta funkcja jest prywatna (w zasadzie prywatnawa),
    wykonuje właściwe obliczenia.
    """
    a, b = 0, 1
    while True:
        a, b = b, a + b
        yield b

def fib(n):
    """
    Zwraca N-tą liczbę Fibonacciego
    """
    r = 0
    for ii, r in zip(range(n), _fib_generator()):
        pass
    return r


def fib2(n): # return Fibonacci series up to n
    result = []
    r = 0
    for r in _fib_generator():
        if r > n:
            return result
        result.append(r)
```

In [2]:
```
fib(3)
```

Out[2]: 3

In [3]:
```
fib2(2)
```

Out[3]: [1, 2]

In [11]:
```
%%bash

 ls *.py
```

```
fib.py
```

1.1 Importowanie

```
In [12]: import fib # Zaimportowałem cały moduł

In [13]: fib.fib(30) # By wykonać funkcję muszę podać nazwę modułu o nazwę obiektu wewnątrz modułu

Out[13]: 1346269

In [14]: if 'fib2' in globals():
             del fib2 # Usuwamy funkcję fib2

In [16]: from fib import fib, fib2 #Importujemy poszczególne funkcje z modułu

In [17]: fib2(3)

Out[17]: [1, 2, 3]

In [18]: if '_fib_generator' in globals():
             del _fib_generator # Usuwamy funkcję fib2

In [19]: from fib import * # Importujemy wszystkie nie prywatne funkcje z modułu

In [20]: _fib_generator # Jest prywatne więc * go nie zaimportowała


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-20-0fbd5aa994dc> in <module>()
    ----> 1 _fib_generator # Jest prywatne więc * go nie zaimportowała


        NameError: name '_fib_generator' is not defined
```

1.2 Moduły są keszowane

Pierwsza instrukcja `import foo` importująca dany plik i zapisuje go w pamięci. Potem następne instrukcje `import foo` ładują dane z pamięci, ewentualne zmiany w pliku nie będą uwzględnione.

Istnieją metody na **przeładowanie** modułów, ale nie jest to bardzo proste narzędzie.

1.3 Import wewnątrz funkcji

Instrukcja `import` może też znaleźć się wewnątrz funkcji, wtedy zaimportowana nazwa jest widoczna tylko w tej funkcji

```
In [21]: if 'fib2' in globals():
             del fib2 # Usuwamy funkcję fib2

In [22]: fib2(3)


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-22-bfadab956885> in <module>()
```

```
    ----> 1 fib2(3)


        NameError: name 'fib2' is not defined
```

```
In [23]: def foo():
             from fib import fib, fib2 #Importujemy poszczególne funkcje z modułu
             return fib2(4)
         foo()
```

```
Out[23]: [1, 2, 3]
```

```
In [24]: fib2(3)
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-24-bfadab956885> in <module>()
    ----> 1 fib2(3)


        NameError: name 'fib2' is not defined
```

1.4 Skąd moduły są importowane

Maszyneria importu Pythona jest dość skomplikowana i wspiera również dynamiczne, leniwe generowanie nowych modułów. W "zwykłym przypadku" (nie spotkałem jeszcze niezwykłego) moduły importowane są z:

- Moduły wbudowane.
- Z paczek systemowych. O tym jak zainstalować dodatkowe moduły pythona będzie później. Z tego miejsca importowany jest kod kiedy piszecie: `import math`.
- Z katalogu w którym znajduje się główny moduł. Czyli jeśli mam w katalogu dwa pliki `foo.py` oraz `bar.py` i wywołam `python foo.py` to plik ten będzie mógł wykonać `import bar`.
- Z katalogów znajdujących się w zmiennej `PYTHONPATH`. Katalogi w tej zmiennej przeszukiwane są po kolei.

1.5 Deklaracja kodawania

Pliki źródłowe Pythona są plikami w tekstowymi i jako takie mogą być zapisane w dowolnym kodowaniu. By wyspecyfikować kodowanie pliku źródłowego można posłużyć się komentarzem:

```
# -*- coding: utf-8 -*-
```

Musi on być pierwszą lub drugą linią pliku

1.6 Paczki

Paczka jest zgrupwaniem modułów, moduły te **w zasadzie** nie są w żaden sposób powiązane silniej niż gdyby nie były w jednej paczce.

Powiedzmy że chcę zrobić paczkę zawierającą narzędzia do generowania różnych matematycznych serii.

Paczka wygląda tak:

In [26]:
```
%%bash

ls series
```

```
fibonacci.py
__init__.py
primes.py
_utils.py
```

1.7 Zawartość pliku _utils.py

In [27]:
```
# -*- coding: utf-8 -*-

class AbstractSeries(object):

    def _series_generator(self):
        pass

    def get_series_element(self, N):
        assert N > 0
        for ii, element in zip(range(N), self._series_generator()):
            pass
        return element

    def get_series_elements_less_than(self, x):
        result = []

        for elem in self._series_generator():
            if elem > x:
                break
            result.append(elem)

        return result

    def get_n_series_elements(self, N):
        result = []

        for ii, element in zip(range(N), self._series_generator()):
            result.append(element)

        return result
```

1.8 Zawartość pliku _fibonacci.py

```
# -*- coding: utf-8 -*-

from . import _utils # Zwróćie uwagę na relatywny import!


class Fibonacci(_utils.AbstractSeries):
     def _series_generator(self):
        a, b = 0, 1
        while True:
            a, b = b, a + b
            yield b
```

1.9 Importy a paczki

Instrukcje import wymagają podania absolutnej nazwy modułu który importujemy
`from series import _utils`
`from series._utils import AbstractSeries`
Możliwe są importy relatywne do aktualnego modułu:
`from . import _utils # Z tego samego katalogu`
`from .. import fib # Z katalogu wyżej`

1.10 Import as

Możliwe jest przezwanie zaimportowanego obiektu np:

```
In [28]: from fib import fib as fibonacci

In [29]: fibonacci(10)

Out[29]: 89
```

1.11 Problemy z importami

Tutaj wchodzimy ta taki trochę bagnisty grunt. Rozważmy taką strukturę katalogów
Generalnie instrukcja

```
import foo
```

importuje moduł w sposób absolutny jeśli mamy taki układ paczek:

- `foo.py`
- Zawartością tego pliku jest: `print('foo')`
- `main.py`
- Zawartością tego pliku jest: `from bar import baz`
- `bar`
- `__init__.py`
- `foo.py`
- Zawartością tego pliku jest: `print('Gotha!')`
- `baz.py`
- Zawartością tego pliku jest `import foo`

Polecenie `python main.py` spowoduje wyświetlenie napisu `foo`, natomiast polecenie `python bar/baz.py` spowoduje wyświetlenie `Gotha!`

1.12 Circular imports

Powiedzmy że mamy taką paczke:

```
In [30]: %%bash
         rm -rf recursive_import/__pycache__
         rm recursive_import/*pyc

rm: cannot remove ‘recursive_import/*pyc’: No such file or directory

In [34]: %%bash

         ls recursive_import
```

```
__init__.py
rec_a.py
rec_b.py
```

```
In [35]: %%bash
         cat recursive_import/rec_a.py
```

```
# -*- coding: utf-8 -*-

from . import rec_b

class RecA:
    def __init__(self):
        rec_b.function(self)
```

```
In [36]: %%bash
         cat recursive_import/rec_b.py
```

```
# -*- coding: utf-8 -*-

from .rec_a import RecA

def function(a):
    assert isinstance(a, RecA)
```

Moduły te wzajemnie się importują. W zasadzie nie jest to problem, ale...

```
In [37]: %%bash

         python -c 'import recursive_import.rec_a'
```

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/home/jb/dydaktyka/web-aplikacje/content/static/zaj3/recursive_import/rec_a.py", line 3, in <mod
    from . import rec_b
  File "/home/jb/dydaktyka/web-aplikacje/content/static/zaj3/recursive_import/rec_b.py", line 3, in <mod
    from .rec_a import RecA
ImportError: cannot import name 'RecA'
```

```
In [38]: %%bash

         python -c 'import recursive_import.rec_b'
```

1.13 Circular imports

Jak sobie z tym radzić?

- Nie tworzyć cyklicznych odwołań przy importowaniu, jest to znak **jakichś** problemów z kodem.
- Czasem można zaimportować odpowiednie symbole wewnątrz funkcji (jeśli nie są często używane)
- Są też inne narzędzia

1.14 Funkcja main

- Generalnie w Pythonie nie ma czegoś takiego jak funkcja `main`
- Cały kod który zawarty jest w module zostanie wykonany kiedy wykonamy (`python module`) lub zaimportujemy (`import module`) moduł.
- Czasem nie chcemy wykonywać części kodu podczas importu.

```
In [39]: def main():
             print("Hello World!")
         if __name__ == "__main__":
             main()
```

```
Hello World!
```

```
In [ ]:
```