



# PYTHON

TABLICE INFORMATYCZNE • Adalbert Arsen



## 1. SKRYPTY PYTHON

### Wykonywanie skryptów

|                                                                                                                                                                                                                                   |                             |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| Uruchomienie z powłoki graficznej, np. IDLE:                                                                                                                                                                                      | <code>otwórz plik+F5</code> |
| Uruchomienie z powłoki systemu:                                                                                                                                                                                                   | <code>python plik.py</code> |
| W systemach unixowych można pominąć interpreter <code>python</code> , jeżeli informacje o nim, poprzedzone shebangiem ( <code>#!</code> ), znajdują się w nagłówku skryptu. Taki skrypt musi też mieć uprawnienia do wykonywania: | <code>./plik.py</code>      |

### Pliki Python

|                                        |                                                                 |
|----------------------------------------|-----------------------------------------------------------------|
| <code>.py</code>                       | skrypty, które możemy modyfikować                               |
| <code>.pyc</code> , <code>.pyo</code>  | skrypty skompilowane (nigdy nie należy ich edytować ręcznie)    |
| <code>.pyz</code> , <code>.pywz</code> | skompresowane archiwum skryptów                                 |
| <code>.pyd</code>                      | odpowiednik biblioteki dynamicznej DLL Windows                  |
| <code>.ipynb</code>                    | pliki IPython Notebook (Jupyter Notebook)                       |
| <code>.pyw</code>                      | skrypty wykonywane za pomocą <code>pythonw.exe</code> (Windows) |

### Przykładowa struktura pliku .py

|                                                   |                                                      |
|---------------------------------------------------|------------------------------------------------------|
| Ścieżka do interpretera (Unix, opcjonalnie), np.: | <code># !/usr/bin/env python</code>                  |
| Kodowanie znaków (opcjonalnie), np.:              | <code># -*- coding=utf-8 -*-</code>                  |
| Import bibliotek, np. NumPy:                      | <code>import numpy as np</code>                      |
| Definicje funkcji, procedur, klas:                | <code>def komentarz():<br/>    print('Witam')</code> |
| Warunek sekcji <code>__main__</code> :            | <code>if __name__ == '__main__':</code>              |
| Główna część programu:                            | <code># pozostałe instrukcje</code>                  |

Zmienna prywatna `__name__` przyjmuje wartość `__main__` tylko wtedy, gdy uruchamiamy skrypt jako samodzielny program. Wstawienie powyższego warunku pozwoli nam importować z pliku funkcje i klasy bez wykonywania programu głównego znajdującego się w sekcji `__main__`.

### Kodowanie znaków

Strona kodowa pozwala edytowemu odczytać znaki zapisane w plikach `.py`. Informacje o stronie kodowej umieszczamy tak, jak to pokazano w przykładzie powyżej.

```
# -*- coding: kodowanie -*-
```

Zalecany sposób. Należy użyć `utf-8`, a gdy nie ma takiej możliwości, polskie znaki są zapewniane przez `iso-8859-2` lub `cp1250`.

```
# coding: kodowanie
```

Alternatywa.

Niektóre edytory mogą nie rozpoznawać powyższego zapisu. Konsola interaktywna może mieć inną stronę kodową. Aby sprawdzić stronę kodową, należy wykonać:

```
import sys; print(sys.stdin.encoding)
```

## 2. KOMENTARZE

Komentarze jednoliniowe poprzedza się minimum jednym znakiem `#`:

```
# wykomentować można całą linię  
print("ok") # lub umieścić komentarz po instrukcji  
Komentarze blokowe otwiera się i zamyka trzema cudzysłowami ('''') lub trzema apostrofami (''''):  
''' komentarz blokowego można użyć do wykomentowania  
jednej, dwóch lub większej liczby linii'''
```

Komentarz blokowy, potocznie zwany **docstring**, umieszczony na początku funkcji (lub modułu) będzie dostępny przez metodę `nazwaFunkcji.__doc__` lub `nazwaModułu.__doc__`. Komentarze blokowe mogą posłużyć do automatycznego wygenerowania dokumentacji w pliku `.html`. Pozwalają na to takie projekty jak: `pydoc`, `epydoc` lub `sphinx-doc`.

## 3. IMPORT MODUŁÓW

Aby móc korzystać z funkcji danego modułu, należy go najpierw zaimportować.

| Składnia                                                                                                      | Przykład                                                  |
|---------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| <code># z modułu importujemy jedną wybraną funkcję</code><br><code>from nazwaModułu import nazwaMetody</code> | <code>from math import sqrt<br/>b = sqrt(4)</code>        |
| <code># import wszystkich funkcji z modułu</code><br><code>from nazwaModułu import *</code>                   | <code>from math import *<br/>b = sqrt(4)</code>           |
| <code># import całego modułu</code><br><code>import nazwaModułu</code>                                        | <code>import math<br/>b = math.sqrt(4)</code>             |
| <code># import całego modułu pod aliasem</code><br><code>import nazwaModułu as jakiśAlias</code>              | <code>import math as blabla<br/>b = blabla.sqrt(4)</code> |

Zapis `from nazwaModułu import *` może być źródłem konfliktów, powinno się go unikać.

## 4. TWORZENIE WŁASNYCH MODUŁÓW

Umieszczenie pustego pliku o nazwie `__init__.py` w wybranym katalogu pozwoli Pythonowi zidentyfikować dany katalog jako moduł i spowoduje między innymi rekurencyjne przeszukiwanie podkatalogów. Aby katalog (tzn. moduł) był widoczny, musi znajdować się w jednej ze ścieżek `sys.path`.

Nazewnictwo: zmiennym oraz skryptom `.py` nigdy nie należy nadawać nazw, które zostały już nadane funkcjom. Trzeba też unikać nazw zastrzeżonych i słów kluczowych.

```
katalog/  
__init__.py  
funkcje.py  
podkatalog/  
__init__.py  
macierze.py  
wykresy.py
```

### Co może znajdować się w pliku \_\_init\_\_.py

Zmienna `__all__` typu `list` zawiera listę podrzędnych modułów, które będą importowane przy użyciu składni `from nazwaModułu import *`. W poniższym przykładzie, aby `macierze.py` i `wykresy.py`, znajdujące się w podkatalogu, importowały się przy użyciu `from katalog import *`, w pliku `__init__.py` należy zadeklarować:

```
__all__ = ['macierze', 'wykresy']
```

W Python 2 znaki Unicode nie są dopuszczalne w `__all__`. Rozwiązanie tego problemu jest następujące:

```
__all__ = [n.encode('ascii') for n in __all_] # tylko dla Python 2
```

Zmienna tekstowa `__version__` może zawierać dowolne informacje o wersji.

Zmienna tekstowa `__author__` powinna zawierać informacje o autorze.

W nagłówku pliku `__init__.py` można też umieścić komentarz `docstring`.

## 5. SŁOWA KLUCZOWE — NAZWY ZASTRZEŻONE

Listę słów kluczowych można otrzymać w następujący sposób:

```
import keyword  
print(keyword.kwlist)
```

Listę nazw zastrzeżonych obejmującą wszystkie nazwy funkcji typów wbudowanych można otrzymać w następujący sposób:

| Python 2                                             | Python 3                                       |
|------------------------------------------------------|------------------------------------------------|
| <code>import __builtin__<br/>dir(__builtin__)</code> | <code>import builtins<br/>dir(builtins)</code> |

## 6. TYPY DANYCH

### Typy arytmetyczne

#### int

Python 2: wartości całkowitoliczbowe typu `int` mieszczą się w zakresie od  $+2\,147\,483\,647$  do  $-2\,147\,483\,648$ . Wartości całkowitoliczbowe wychodzące poza zakres `int` kończymy `L`, np. `a = 4721885298529L`. Jeżeli jakaś wartość przekroczy zakres `int`, Python 2 automatycznie zmieni typ na `long`. Python 3: typ `int` jest praktycznie typem `long`. Nie wolno podawać kończącego `L`. Liczby `long` mogą być zapisane w postaci heksadecymalnej lub oktalnej, np. `a = 44B664BBF61`.

#### float

Liczby rzeczywiste deklarujemy przy pomocy kropki dziesiętnej. Dopuszczalne są też zapisy na bazie `e` lub `E` z podaniem wykładnika, np. zapis `a = 2.5e2` odpowiada wyrażeniu  $2.5 \cdot 10^2$ , które jest tożsame z `a = 250`. lub `a = 250.0`.

#### complex

Liczby urojone zapisuje się, podając obowiązkowo część rzeczywistą oraz część urojoną `j`, np. `a = 25+3j`.

Do konwersji pomiędzy typami służą funkcje `int()`, `float()`, `long()` i `complex()`. Dodatkowe informacje można uzyskać po zaimportowaniu biblioteki `sys` lub z funkcji `type()`:

```
sys.maxint _____ Maksymalny zakres int w Python 2.  
sys.maxsize _____ Maksymalny zakres int w Python 3.  
type(a) _____ Zwraca typ zmiennej a.  
sys.getsizeof(a) _____ Rozmiar obiektu a w bajtach.
```

W Pythonie wszystkie zmienne są obiektami. Liczby też. Każdy obiekt posiada od kilku do kilkunastu metod własnych dostępnych „po kropce”. Jeżeli zmienna `a` jest typu liczbowego, będą to metody usprawniające pracę z liczbami. Np. dla `a = 2.0` metoda `a.hex()` zwróci nam postać heksadecymalną liczby. Jeżeli `a` jest łańcuchem znaków, będą to metody usprawniające pracę ze znakami, np. `a.title()` zamieni małe litery na duże. Każda zmienna liczbowa posiada metodę `.bit_length()` zwracającą rozmiar w bajtach.

### Typ logiczny (bool)

Dopuszczalne wartości zmiennej typu logicznego to `True` lub `False` (wielkość liter ma znaczenie). Liczba całkowita 1 może być interpretowana jako logiczne `True`, a liczba 0 jako `False`.

```
a = True ; b = 1  
if a and b:  
    print("ok")
```

### Znaki (str)

Python 2: znaki są typu ASCII. Deklarację znaków Unicode poprzedzamy `u`. Python 3: znaki są typu Unicode, pomijamy `u`:

```
ptak = u"gżegżółka" # Python 2  
ptak = "gżegżółka" # Python 3
```

*Dalsza część książki dostępna w wersji  
pełnej.*

