PYTHON

TABLICE INFORMATYCZNE • Adalbert Arsen

Helion

1. SKRYPTY PYTHON

Wykonywanie skryptów

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

Pliki Python

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

Przykładowa struktura pliku .py

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

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 edytorowi tekstowemu 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 jednowierszowe poprzedza się minimum jednym znakiem `#`:

```
# wykomentować można całą linijkę
print("ok")  # lub umieścić komentarz po instrukcji
```

Komentarze blokowe otwiera się i zamyka trzema cudzysłowami (`"""`) lub trzema apostrofami (`'''`):

```
""" komentarza blokowego można użyć do wykomentowania
jednej, dwóch lub większej liczby linijek"""
```

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
# z modułu importujemy jedną wybraną funkcję from *nazwaModułu* import *nazwaMetody*	`from math import sqrt` `b = sqrt(4)`
# import wszystkich funkcji z modułu from *nazwaModułu* import *	`from math import *` `b = sqrt(4)`
# import całego modułu import *nazwaModułu*	`import math` `b = math.sqrt(4)`
# import całego modułu pod aliasem import *nazwaModułu* as *jakiśAlias*	`import math as blabla` `b = blabla.sqrt(4)`

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 przeszukanie 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 `form 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
`import __builtin__` `dir(__builtin__)`	`import builtins` `dir(builtins)`

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
```