Python - podstawy

Dr inż. Maciej Miłostan, Instytut Informatyki,
Politechnika Poznańska

Co to jest Python

- Język programowania
- Interpretowano-kompilowany
- Elastyczny
- Łatwy do nauki
- O bogatej funkcjonalności
- Posiada wydajne wysoko poziomowe struktury danych
- Proste, ale efektywne podejście do programowania zorientowanego obiektowo
- Elegancka składni, dynamiczne typowanie czynią i interpretowany charakter cynią go idealnym w RAD i do pisania skryptów
- Open source, wiele bibliotek standardowych

Co to jest Python

- Posiada interfejsy do wielu języków programowania
- Bardzo łatwo można rozbudować interpreter o funkcje i typy zaimplementowane w C lub C++
- Uwaga w użyciu są dwie niekompatybilne wersje języka:

  2.x i 3.x
- Będziemy mówić głównie o wersji 3 (za wyjątkiem BioPython-a)
- Nazwa pochodzi od “Monty Python’s Flying Circus” i nie ma nic wspólnego z gadami ☺

Linki do dokumentacji

- Pyton – oficjalna strona
- Python v3.2.2 documentation
- Tutorial
- Applications
- *The Python Standard Library*
- *The Python Language Reference*
- *Extending and Embedding the Python Interpreter*

Głośne przypadki użycia

- NASA
  http://www.python.org/about/success/usa/
- Google Inc.
- YouTube.com
- Thawte Consulting

http://www.python.org/about/quotes/

*Źródło: An Introduction to Perl Programming, Dave Cross, Magnum Solutions Ltd

Filozofia Pythona

- The Zen Of Python

```
mily@cerber:~> python
Python 2.6.2 (r262:71600, Jun 17 2010, 13:37:45)
[GCC 4.4.1 [gcc-4_4-branch revision 150839]] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
```

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

Skąd pobrać?

- Jest dystrybuowany z większością dystrybucji linuksowych, jeśli używasz linuksa lub Mac OS-a, to prawdopodobnie już go masz
- Oficjalna strona Pythona
  - http://www.python.org/download/
- Wersje pod windows też tam są
- Najnowsze wersje Pythona to 2.7.2 i 3.2.2

Interpreter Pythona

- Zwykle jest instalowany w:

/usr/local/bin/python3.2 lub /usr/bin/...

W Windows: C:\Python32

- Wywołanie konsoli:

python3.2

- CTRL-D lub CTR-Z (Windows) kończy działanie interpretera
- Hisoria i edycja CTRL-P, C-N, C-A, C-F, C-B, C-_, C-K, C-Y

Interpreter pythona

- python –c komenda [arg]
- python –m moduł [arg]
- Dostęp do argumentów

import sys

sys.argv

sys.argv[0] jest ustawiane na '-`

W przypadku *-c command*, sys.argv[0] jest ustawiane na '-c`

-m module to sys.argv[0] jest ustawione na nazwę modułu

Tryb interaktywny

```
$ python3.2
Python 3.2 (py3k, Sep 12 2007, 12:21:02)
[GCC 3.4.6 20060404 (Red Hat 3.4.6-8)] on linux2
Type "help", "copyright", "credits" or "license" for more
    information.
>>>
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

Tu jest obowiązkowe wcięcie

Pusty wiersz jest potrzebny, żeby zaznaczyć koniec bloku istrukcji

Tryb interaktywny

```
$ python3.2
Python 3.2
[GCC 3.4.6                                                    2
Type "help                                                    more
    informat
>>>
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

Zwróć uwagę na brak nawiasów klamrowych, średników, end-ów, begin-ów itp. ozdobników znanych z innych języków!!!

Skrypty wykonywalne

- Znana konwencja – na początku pliku umieszczamy:

#! /usr/bin/env python3.2

#! /usr/bin/python

Oczywiście linia musi się kończyć \n a nie \r\n

- I nadanie praw:

$ chmod +x myscript.py

Strona kodowa kodów ☺

- Domyślnie UTF-8
- Można zmienić – bezpośredni po linii #! umieszczamy:

```
# -*- coding: encoding -*-
```

np.:

```
# -*- coding: cp-1250 -*-
```

Pliki startowe dla trybu interaktywnego

- Zmienna środowiskowa PYTHONSTARTUP (działa na podobnej zasadzie jak .profile, albo .bashrc w przypadku powłoki linuksa)
- Skrypt załaduje się tylko w przypadku wywołania powłoki interaktywnej
- Jeśli chcemy go wywołać również ze skryptu, to:

```
import os
filename = os.environ.get('PYTHONSTARTUP')
if filename and os.path.isfile(filename):
    exec(open(filename).read())
```

Customization modules

- sitecustomize
- usercustomize
- >>> import site
- >>> site.getusersitepackages()
  '/home/user/.local/lib/python3.2/site-packages'
- -s wyłącza automatyczny import

Komentarze

```
# to jest pierwszy komentarz
SPAM = 1                # a to drugi
                        # ... No i jeszcze trzeci!
STRING = "# To nie jest komentarz."
```

Python jako kalkulator

Python 3.x

```
>>> 2+2
4
>>> # This is a comment
... 2+2
4
>>> 2+2  # and a comment on
    the same line as code
4
>>> (50-5*6)/4
5.0
>>> 8/5 # Fractions aren't lost
    when dividing integers
1.6
```

Python 2.x

```
>>> 2+2
4
>>> #To jest komentarz
... 2+2
4



>>> (50-5*6)/4
5
>>> 8/5 #Ułamki są gubione
1
```

Python jako kalkulator

Python 3.x

```
>>> # Integer division
    returns the floor:
... 7//3
2
>>> 7//-3
-3
```

Python 2.x

```
>>> 7/3
2
>>> 7.0/3
2.3333333333333335
>>> 7.0//3
2.0
>>> 7.0//-3
-3.0
>>> 7//-3
-3
```

Przypisywanie wartości zmiennym

- Pojedyncze przypisania

```
>>> width = 20
>>> height = 5*9
>>> width * height
900
```

- Przypisania równoległe

```
>>> x = y = z = 0  # Zero x, y and z
>>> x
0
>>> y
0
>>> z
0
```

Zmienne muszą być definiowane (inicjowane)

```
>>> # spróbuj użyć niezdefiniowanej
    zmiennej
... n
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Operacje zmienno-przecinkowe

- Pełne wsparcie dla operacji zmiennoprzecinkowych – operacje z liczbami typu integer są konwertowane na float w przypadku, gdy którykolwiek argument jest zmiennoprzecinkowy (float)

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

Liczby zespolone

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0, 1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

- Zawsze są reprezentowane jako dwa float-y

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Konwersje typów

- chr() wartość całkowitą na znak o odpowiednim kodzie ASCII
- complex() wartość całkowitą, rzeczywistą lub napisową na zespoloną
- ord() znak na jego kod ASCII
- float() wartość całkowitą lub napisową na rzeczywistą
- int() wartość rzeczywistą lub napisową na całkowitą
- str() jakąkolwiek wartość na napis
- eval() interpretuje wartość napisową tak, jak Python, i oblicza wynik

Liczby zespolone a konwersje typów

```
>>> a=3.0+4.0j
>>> float(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can't convert complex to float; use abs(z)
>>> a.real
3.0
>>> a.imag
4.0
>>> abs(a)  # sqrt(a.real**2 + a.imag**2)
5.0
```

Python jako kalkulator cd.

- Zmienna _

```
>>> podatek = 12.5 / 100
>>> cena = 100.50
>>> cena * podatek
12.5625
>>> podatek + _
113.0625
>>> round(_, 2)
113.06
```

Łańcuchy znaków

- Cudzysłowie

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

```
hello = "This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
    Note that whitespace at the beginning of the line is\
 significant."

print(hello)
```

```
This is a rather long string containing
several lines of text just as you would do in C.
    Note that whitespace at the beginning of the line is significant.
```

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

produces the following output:

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Surowy łańcuch

- Surowy łańcuch (raw string)

```
hello = r"This is a rather long string containing\n\
several lines of text much as you would do in C."

print(hello)
```

```
This is a rather long string containing\n\
several lines of text much as you would do in C.
```

Operacje na łańcuchach

- Konkatenacja

```
>>> word = 'Help' + 'A'
>>> word
'HelpA‘
```

- Generowanie powtórzeń

```
>>> '<' + word*5 + '>'
'<HelpAHelpAHelpAHelpAHelpA>'
```

Operacje na łańcuchach

- Konkatenacja dwóch łańcuchów

```
>>> 'str' 'ing'                    #  <-  This is ok
'string'
>>> 'str'.strip() + 'ing'   #  <-  This is ok
'string'
>>> 'str'.strip() 'ing'     #  <-  This is invalid
  File "<stdin>", line 1, in ?
    'str'.strip() 'ing'
                  ^
SyntaxError: invalid syntax
```

```
>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[2:4]
'lp`
>>> word[:2] # The first two characters 'He'
>>> word[2:] # Everything except the first two characters 'lpA'
```

Łańcuchy są niemutowalne

```
>>> word[0] = 'x'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'str' object does not support item assignment
>>> word[:1] = 'Splat'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'str' object does not support slice assignment
```

...ale można kopiować ich fragmenty

```
>>> 'x' + word[1:]
'xelpA'
>>> 'Splat' + word[4]
'SplatA`
>>> 'x' + word[1:]
'xelpA'
>>> 'Splat' + word[4]
'SplatA'
```

```
>>> word[:2] + word[2:]
'HelpA'
>>> word[:3] + word[3:]
'HelpA'
```

```
>>> word[:2] + word[2:]
'HelpA'
>>> word[:3] + word[3:]
'HelpA‘

>>> word[1:100]
'elpA'
>>> word[10:]
''
>>> word[2:1]
''
```

```
>>> word[-1] # The last character 'A' >>>
  word[-2] # The last-but-one character 'p'
>>> word[-2:] # The last two characters 'pA'
>>> word[:-2] # Everything except the last
  two characters 'Hel‘


>>> word[-0] # (since -0 equals 0) 'H'
```

```
>>> word[-100:]
'HelpA'
>>> word[-10]    # error
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

```
>>> word[-100:]  #przytnie wartość indeksu
'HelpA'
>>> word[-10] # error
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

```
 +---+---+---+---+---+
 | H | e | l | p | A |
 +---+---+---+---+---+
 0   1   2   3   4   5
-5  -4  -3  -2  -1
```

Uwaga:

przedziały są niedomknięte od góry:

word[1:2] zwróci jedną literę

- >>> s = 'supercalifragilisticexpialidocious'
  >>> len(s) 34

Unicode

Python 3.2

```
>>> w="słowo"
>>> len(w)
5
>>> w
'słowo'
```

Uwaga na encoding terminala:

```
>>> w="s³owo"
  File "<stdin>", line 0

    ^
SyntaxError: 'utf8' codec can't decode byte 0xb3
    in position 4: unexpected code byte
```

Python 2.7

```
>>> w="słowo"
>>> len(w)
6
>>> w
's\xc5\x82owo'
>>> w=u"słowo"
>>> len(w)
5
>>> w
u's\u0142owo'
>>>
```

```
>>> "słowo".encode('utf-8')
b's\xc5\x82owo`

>>> 'Hello\u0020World !'
'Hello World !'
```

Listy

- Listy są mutowalne
- Mogą przechowywać rówież dane złożone np. inne listy

Listy - przykłady

```
>>> a = ['spam', 'eggs', 100, 1234] >>> a
['spam', 'eggs', 100, 1234]

>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

- Kopia listy

```
>>> a[:]
['spam', 'eggs', 100, 1234]
```

- Mutowanie lisy

```
>>> a ['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
```

```
>>> # Replace some items:
... a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
>>> # Remove some:
... a[0:2] = []
>>> a
[123, 1234]
>>> # Insert some:
... a[1:1] = ['bletch', 'xyzzy']
>>> a
[123, 'bletch', 'xyzzy', 1234]
>>> # Insert (a copy of) itself at the beginning
>>> a[:0] = a
>>> a
[123, 'bletch', 'xyzzy', 1234, 123, 'bletch', 'xyzzy', 1234]
>>> # Clear the list: replace all items with an empty list
>>> a[:] = []
>>> a
[]
```

Długość listy

```
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4
```

Listy zagnieżdżone

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3]
>>> p[1][0]
2
```

Rozszerzanie list

```
>>> p[1].append('xtra')
>>> p [1, [2, 3, 'xtra'], 4]
>>> q [2, 3, 'xtra']
```

Przykładowy program

```
>>> # Fibonacci series:
... # the sum of two
    elements defines the
    next
... a, b = 0, 1
>>> while b < 10:
...             print(b)
...             a, b = b, a+b
...
```

```
1
1
2
3
5
8
```

print

```
>>> a, b = 0, 1
>>> while b < 1000:
... print(b, end=',') ... a,
    b = b, a+b
...
    1,1,2,3,5,8,13,21,34,5
    5,89,144,233,377,610
    ,987,
```

```
>>> i = 256*256
>>> print('The value of i is', i)
The value of i is 65536
```

Instrukcje sterujące

```
>>> x = int(input("Please enter an integer: ")) Please
   enter an integer: 42
>>> if x < 0:
...     x = 0
...         print('Negative changed to zero')
... elif x == 0:
...         print('Zero')
... elif x == 1:
...         print('Single')
... else:
...         print('More')
...
```

- More

Pętla for

```
>>> a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print(x, len(x))
...
cat 3
window 6
defenestrate 12
```

Pętla for

- Nie można modyfikować iterowanych elementów – konieczne jest zrobienie kopii

```
>>> for x in a[:]: # zrób kopię listy (slice)
... if len(x) > 6: a.insert(0, x)
...
>>> a
['defenestrate', 'cat', 'window', 'defenestrate']
```

Użycie range

```
>>> for i in range(3):
... print(i)
...
0
1
2
```

range(5, 10)

5 do 9

range(0, 10, 3)

0, 3, 6, 9

range(-10, -100, -30) -10, -40, -70

```
>>> a = ['Mary', 'had', 'a', 'little',
  'lamb']
>>> for i in range(len(a)):
...       print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

Enumerate i items

```
for i, v in enumerate(['tic', 'tac', 'toe']):
... print(i, v)
...
0 tic
1 tac
2 toe
```

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...         print(k, v)
...
gallahad the pure robin the brave
```

```
>>> print(range(10))
range(0, 10)

>>> list(range(5))
[0, 1, 2, 3, 4]
```

```
 for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

break,continue – zachowują się jak w C

else – nowość – wykonuje się, gdy warunek iterowania nie jest już spełniony (wszystkie elementy zostały przeiterowane)

Wyrażenie pass

- pass – jak w grze w karty pasujemy i nie podjmujemy działań, instrukcja pusta
- Używa się jej tam, gdzie ze wzgl. składni musimy coś wpisać np. przy tworzeniu prototypów, lub aktywnym czekaniu:

```
>>> while True:
...         pass # Busy-wait for
    keyboard interrupt (Ctrl+C)
...

>>> class MyEmptyClass:
...         pass
...

>>> def initlog(*args):
... pass # Remember to
    implement this!
...
```

Definiowanie funkcji

```
>>> def fib(n):    # wypisuje ciąg fibonnaciego do liczby n
...     """ To jes dokumentacja funkcji fib"""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Wywolaj funkcje:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

- Funkcja ma własną tabelę symboli (zmienne lokalne), wartości zmiennym globalnym można przypisać jeśli się to explicite zaznaczy

Wywoływanie funkcji

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89

>>> fib(0)
>>> print(fib(0))
None
```

Zwracanie wyniku

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Wartości domyślne

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError('refusenik user')
        print(complaint)
```

Wartości domyślne są ewaluowane w momencie wywołania

```
>>> i = 5
... def f(arg=i):
...   print(arg)
...
... i= 6
... f()
5
```

WAŻNE

- Wartość domyślna jest ewaluowana tylko raz

```
def f(a, L=[]):
    L.append(a)
    return L

print(f(1))
print(f(2))
print(f(3))
```

- Wynik:

[1]
[1, 2]
[1, 2, 3]

```
def f(a, L=None):
    if L is None: L = []
    L.append(a) return L

print(f(1))
print(f(2))
print(f(3))
```

```
[1]
[2]
[3]
```

Wywoływanie funkcji

- slowo_kluczowe = wartość

```
def pmidor(v, s=‘coś sztywnego',
   a=‘leżeć', type='Norweski Błękit'):
.....
pomidor(1000)
pomidor(a=‘stać’, v=10)
pomidror( ‘tysiac’, s=‘bardzo miękkie’)
pomidor(1000,’miękkie’,’stoi’)
```

- pomidor() #brakuje argumentu
- pomidor(v=5, ‘spadł’) #argument bez słowa kluczowego po słowie kluczowym
- pomidor(5, v=10) #zduplikowana wartość
- pomidor(nie=‘zdefiniowana’) #nie zdefiniowne słowo kluczowe

Argumenty na liście i w słowniku

```
def cheeseshop(kind, *arguments,
    **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of",
     kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    keys = sorted(keywords.keys())
    for kw in keys:
        print(kw, ":", keywords[kw])
```

- cheeseshop("Limburger", "It's very runny, sir.", "It's really very, VERY runny, sir.", shopkeeper="Michael Palin", client="John Cleese", sketch="Cheese Shop Sketch")

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir. It's really very,
    VERY runny, sir.
----------------------------------------
client : John Cleese
shopkeeper : Michael Palin s
ketch : Cheese Shop Sketch
```

Argumenty arbitralne

Przykład 1:

```
def write_multiple_items(file, separator,
  *args): file.write(separator.join(args))
```

Przykład 2:

```
>>> def concat(*args, sep="/"):
...     return sep.join(args)
...
>>> concat("earth", "mars", "venus")
'earth/mars/venus'
>>> concat("earth", "mars", "venus",
  sep=".")
'earth.mars.venus'
```

Rozpakowywanie listy argumentów

```
>>> list(range(3, 6)) # normal call with separate arguments
[3, 4, 5]
>>> args = [3, 6]
>>> list(range(*args)) # call with arguments unpacked from a list
[3, 4, 5]
```

Rozpakowywanie argumentów ze słownika

```
>>> def parrot(voltage, state='a stiff',
    action='voom'):
.........
>>> d = {"voltage": "four million", "state":
    "bleedin' demised", "action": "VOOM"}
>>> parrot(**d)
```

Lambda wyrażenia

- Konstrukcja rodem z języków funkcyjnych – umożliwia tworzenie anonimowych funkcji

Przykład:

```
>>> def make_incrementor(n):
...                 return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Dziękuję za uwagę!