

Koło Studentów Informatyki
Uniwersytetu Wrocławskiego



Python

programming is fun again!

Piotr Wasilewski
wasilewski.piotrek (at) gmail.com

Akademickie Stowarzyszenie Informatyczne

Plan wykładu

- Wstęp
- Język
- Zastosowanie
- Przykłady (na żywo)
- Zakończenie

Wstep

Czym jest Python

- Interpretowany język programowania
(możliwa kompilacja do binarki)
- Wieloplatformowy:
 - Linux/UNIX
 - Windows
 - Mac OS
 - Symbian
 - ...



Co go wyróżnia

- Cechy:
 - dynamiczne typowanie
 - automatyczne zarządzanie pamięcią (*garbage collector*)
 - bloki kodu tworzone poprzez wcięcia (czytelność kodu!)
 - kod jest czytelny i intuicyjny, dzięki czemu Python jest łatwy w nauce

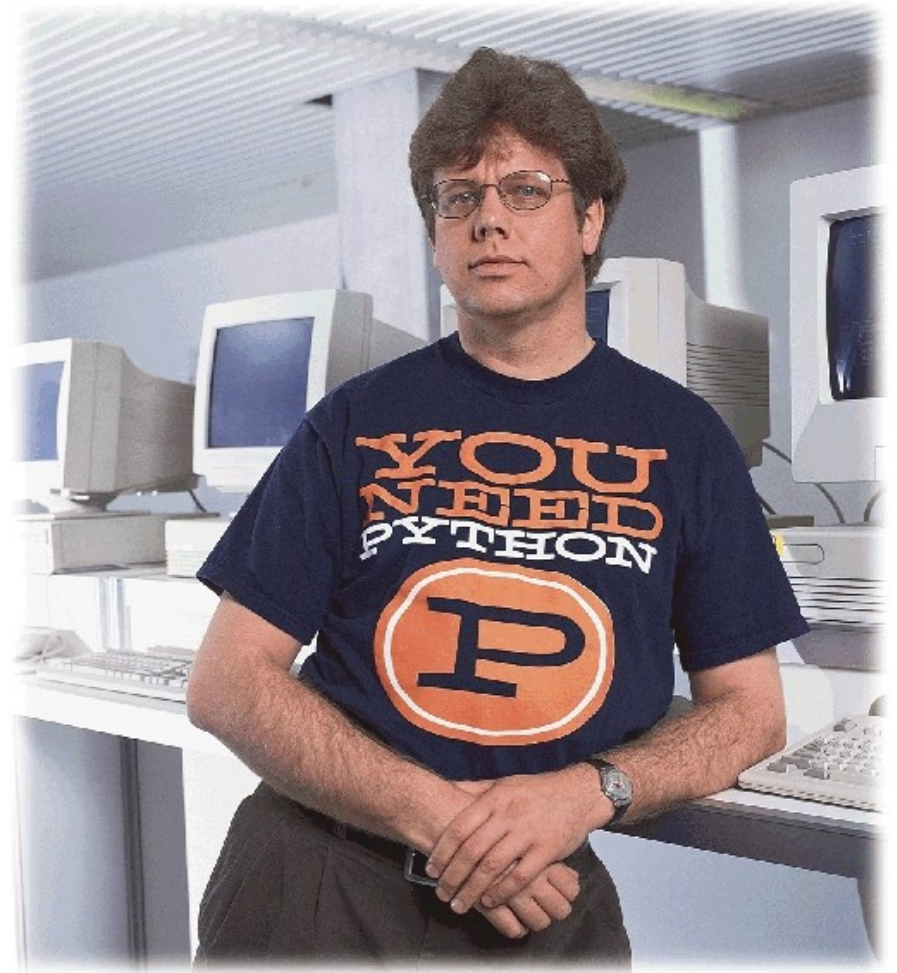
Oblicza Pythona

- CPython – standard, czyli po prostu Python
- Jython – implementacja w Javie
- IronPython - .NET (Mono)
- PyPy – Python w Pythonie
- ChinesePython ;)
- ...



Historia

- Python został stworzony pod koniec lat 80-tych przez Guido van Rossuma
- Wersje stabilne na dzień dzisiejszy to 3.1.2 i 2.6.5



Ale... skąd ta nazwa?!

- Monty Python :)



Jak zacząć?

- Instalacja
 - www.python.org/download/
 - repozytoria
- Edytor tekstu/IDE
 - VIM – oczywiście ;)
 - Gedit, Notatnik, EMACS, ...
 - Eclipse, IDLE, ...
- Do dzieła!

Hello, World!

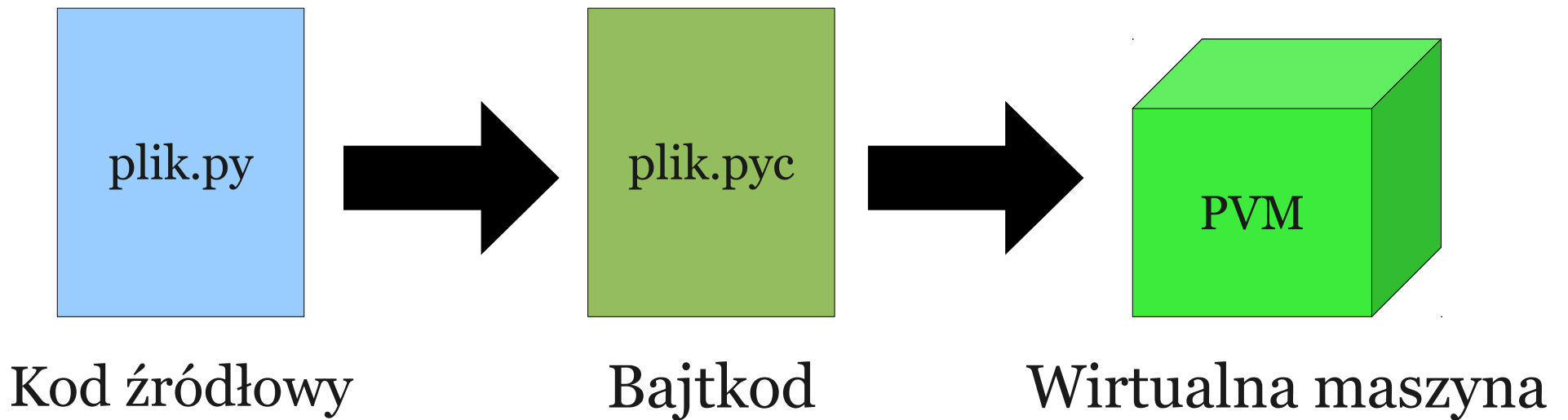
- W sesji interaktywnej:

```
$ python  
>>> print "Hello, World!"  
Hello, World!
```

- W pliku:

```
#!/usr/bin/env python  
print "Hello, World!"
```

Jak działa Python



Język

Typy wbudowane

- string
- int, long, bool
- float
- complex

- listy (list)
- krotki (tuple)
- słowniki (dict)
- zbiory (set)

- pliki (file)

- None



Zmienna – jak utworzyć

```
▪ s = "mielonka na obiad"  
i = 4  
x = 26.654  
l = [3, 6, 8, 9]  
t = ('a', 'b', 'c')  
d = {2: "silly", 5: "walks"}  
...
```

Łańcuchy znaków – typ *string*

- `'ala ma kota'`
- `"tekst\n\tsformatowany"`
- `r'surowy lancuch znakow'`
- `u"unicode aśćź",`
- `" " "wielolinijkowy
tekst" " "`
- `str(14)` `#'14'`

Łańcuchy znaków - operacje

- `len("tekst")` *# 5*
- `"konkatenacja" + r' ' + u"stringów"`
- `a = "tekst"`
`a[1]` *# 'e'*
`a.upper()` *# 'TEKST'*
`a.replace('t', '_')` *# '_eks_'*
- `b = "ala.ma.kota"`
`b.split('.')` *# ['ala', 'ma', 'kota']*

Łańcuchy znaków - wycinki

- `s = "hiszpańska inkwizycja"`

`s[2:7]` *# 'szpan'*

`s[:7]` *# 'hiszpan'*

`s[11:]` *# 'inkwizycja'*

`s[:]` *# 'hiszpańska inkwizycja'*

`s[::-1]` *# 'ajczyiwkni aksnapzsih'*

`s[::3]` *# 'hznanic'*

Łańcuchy znaków – formatowanie tekstu

▪ `"wynik: %f" % 6.9` *# 'wynik: 6.9'*

`"%s %f" % ("spam", 69)` *# 'spam 69'*

▪ `x = 5`

`print "liczba = %i" % x`

Liczby całkowite – typy *int*, *long* i *bool*

- `x = 5`
`y = 3`
`x + y` # 8
`x * 2` # 10
`y ** 3` # 27
`int('123')` # 123
- `bool(69)` # *True*
`bool(0)` # *False*

Liczby zmiennoprzecinkowe – typ *float*

- $f = 3.14$

$f / 2$ *# 1.57*

- Precyzja zależy od systemu (najczęściej podwójna precyzja)

Liczby zespolone – typ *complex*

$$\blacksquare x = 3j + 4$$

$$y = 8j$$

$$x + y \qquad \# 4 + 11j$$

$$x * 2 \qquad \# 8 + 6j$$

Listy – typ *list*

- `l = ['spam', 69, 4+5j]` *#dowolny typ*
- `l.append(4)` *#['spam', 69, 4+5j, 4]*
- `l.pop()` *#ucina ostatni element i zwraca jego wartość*
- `l[1]` *# 69*
- `l[: 2]` *# ['spam', 69]*
- Listy mogą zawierać obiekty dowolnego typu
- Można je dowolnie zagnieżdżać

Krotki – typ *tuple*

- `t = ('parrot', 123, 'spam')`

`t[1:]` *# (123, 'spam')*

- Krotki są obiektami stałymi (nie do końca, ale o tym później...)
- Tak samo jak listy, mogą zawierać obiekty dowolnego typu
- Tak samo można też krotki zagnieżdżać

Słowniki – typ *dict*

- `d = {3: 'spam', 'parrot': 'ni'}`
`d.keys()` *# [3, 'parrot']*
`d.values()` *# ['spam', 'ni']*
`d['nowy'] = 1` *# {3: 'spam', 'nowy': 1, 'parrot': 'ni'}*
`d.pop(3)` *# usuwa parę klucz-wartość*

dla podanego klucza i zwraca wartość

Zbiory – typ *set*

▪ `A = set([1, 44, 5, 13])`

`B = set([1, 32, 5, 13, 6])`

`A.intersection(B)` *# set([1, 5, 13])*

`A.union(B)` *# set([32, 1, 5, 6, 44, 13])*

Pliki – typ *file*

- `f = open('spam.txt', 'r')`
`f.read()` *# odczytuje cały plik*
`f.readline()` *# odczytuje kolejną linię*
`f.write('tekst')` *# dopisuje do pliku 'tekst'*
- Możliwa jest praca w trybie „binarnym”

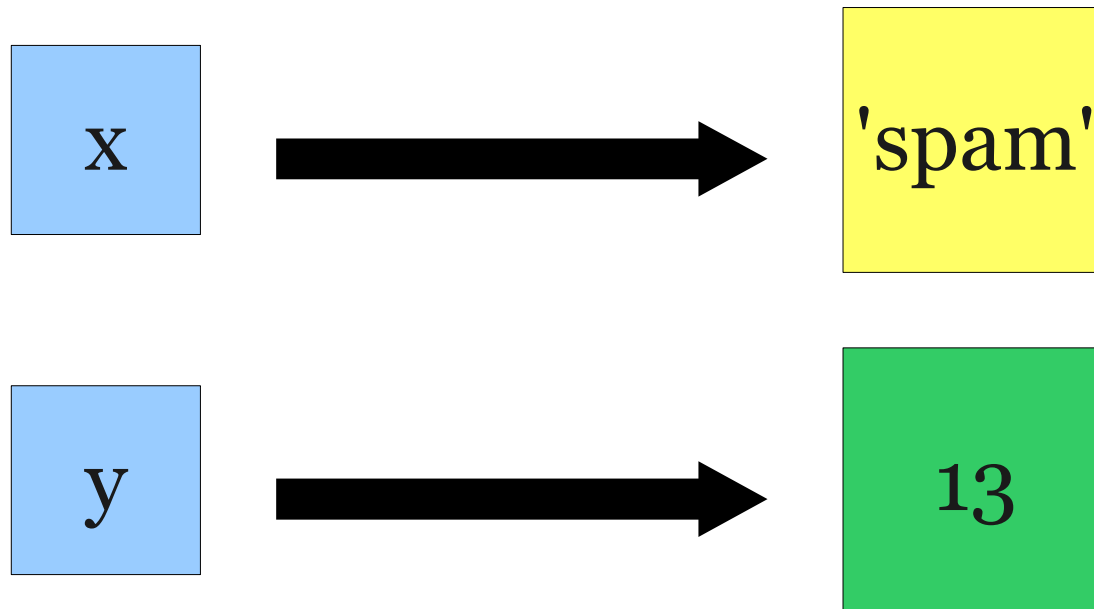
Dynamiczne typowanie

- Nie istnieją zmienne jako takie – operujemy na referencjach do obiektów w pamięci
- Zmieniają się referencje a nie wartości
- Przypisanie tworzy referencję do obiektu
- Jeżeli do obiektu nie odnoszą się żadne referencje, jest on usuwany (*garbage collector*)

Dynamiczne typowanie

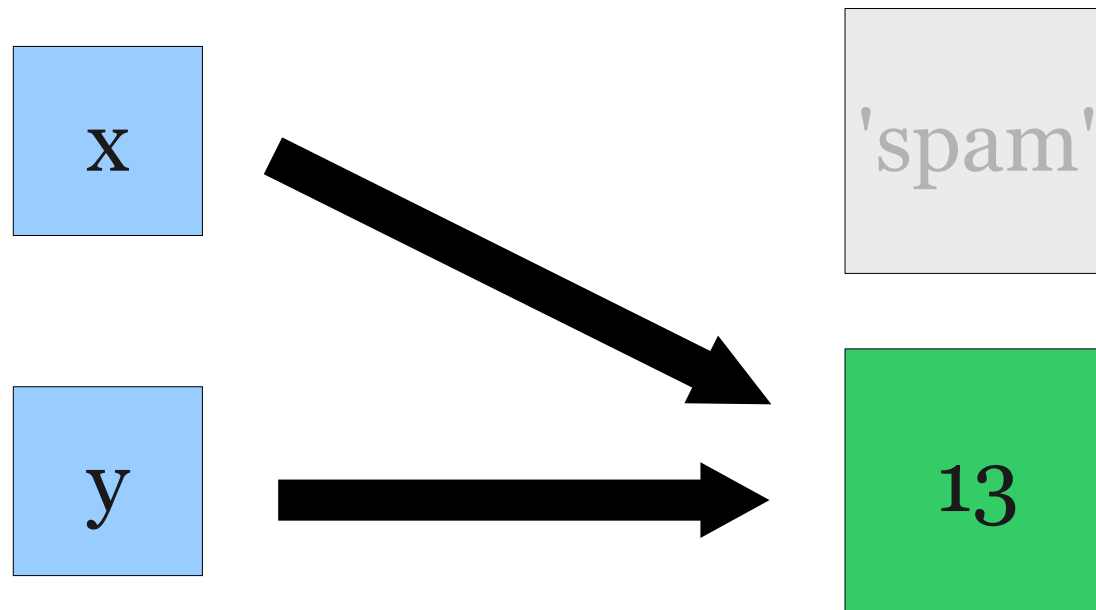
- `x = 'spam'`

`y = 13`



Dynamiczne typowanie

- $x = y$



Klasyfikacja typów

- Typy zmienne (*modyfikowalne w miejscu*):
 - listy
 - słowniki
- Typy niezmiennie:
 - liczby
 - łańcuchy znaków
 - krotki

Klasyfikacja typów

- Tylko obiekty zmienne możemy modyfikować, zachowując je pod tym samym adresem (referencje są nadal aktualne!)
- Chcąc zmienić obiekt typu niezmiennego, musimy przypisać mu nową wartość, np.

```
x = "mielonka"
```

```
x = "smaczna " + x           # rozszerzamy łańcuch
```

Klasyfikacja typów

- Uwaga! Jeżeli w sekwencji umieścimy obiekt typu zmiennego, to modyfikując ten obiekt, modyfikujemy również sekwencję (pośrednio)

```
l = [1, 2]
```

```
t = (l, 7)
```

```
l.append('x')
```

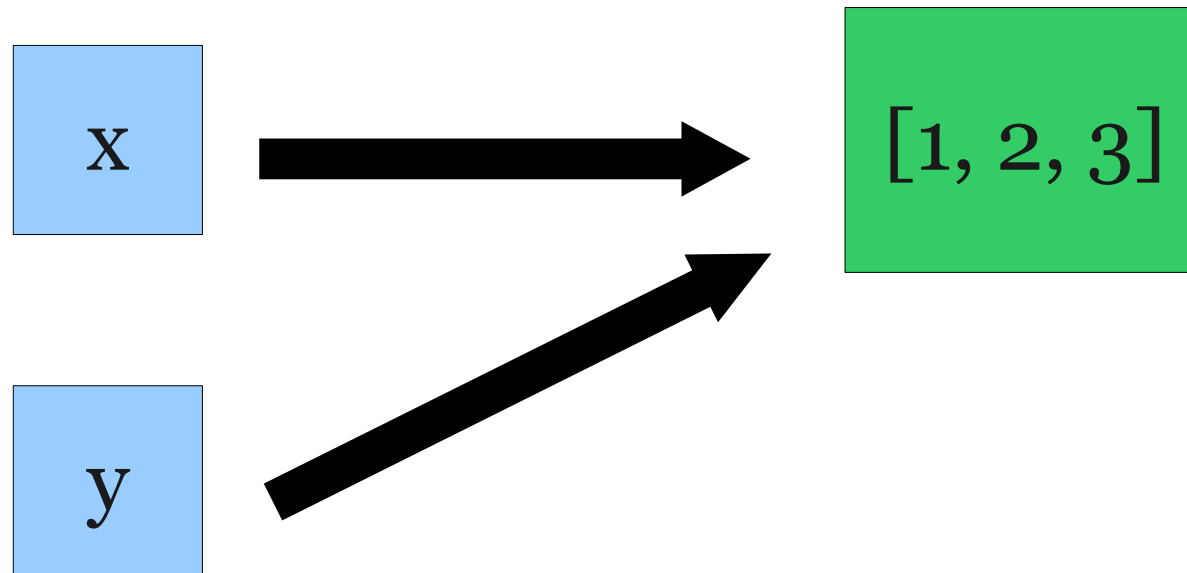
```
# t = ([1, 2], 7)
```

```
# teraz t = ([1, 2, 'x'], 7)
```

Dynamiczne typowanie

▪ $x = [1, 2, 3]$

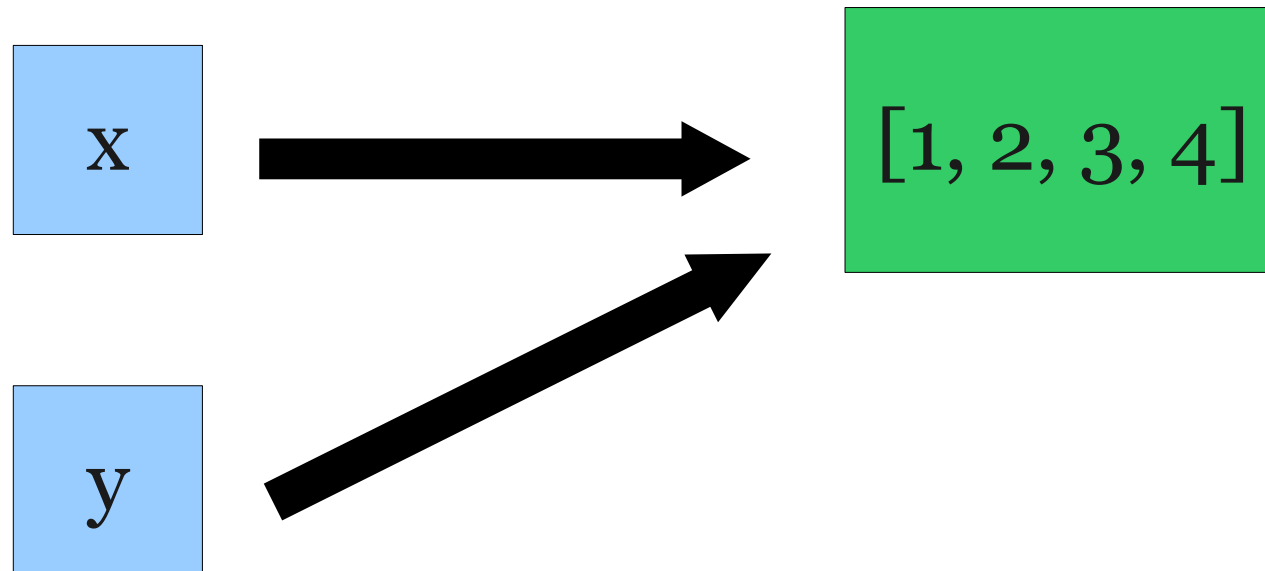
$y = x$



Dynamiczne typowanie

- Uwaga!

```
x.append(4)
```



Instrukcja warunkowa

- Struktura:

if warunek:

 instrukcje

elif warunek2:

 instrukcje

(...)

else:

 instrukcje

Instrukcja warunkowa

- Warunek jest wyrażeniem a nie instrukcją!
(czyli np. nie może zawierać przypisania)
- Pisząc warunek staramy się nie używać nawiasów – chodzi o czytelność kodu
- Instrukcja kończy się dwukropkiem!

Parę słów o strukturze kodu

- Bloki kodu wydzielamy robiąc wcięcia
- „Głębokość” wcięć nie ma znaczenia, ale musimy być konsekwentni
- Nie musimy używać średników (ale możemy)
- Chcąc stworzyć pusty blok stosujemy `pass` (wbrew pozorom to przydatne)

Pętla *while*

- Struktura:

while warunek:

 instrukcje

else:

 instrukcje

- Instrukcje: `break` oraz `continue`
- Blok *else* jest wykonywany, jeżeli pętli nie zakończyła instrukcja *break*

Pętla *for*

- Struktura:

for element **in** sekwencja:

instrukcje

else:

instrukcje

- Pętla *for* wykorzystuje iterację do przejścia po elementach danej sekwencji
- Blok *else* działa tak samo jak w pętli *while*

Listy składane

- Składnia:

```
x = [wyrażenie for elem in sekw]
```

- Tworzy nową listę, wykonując wyrażenie dla każdego elementu z sekwencji

```
lista = [1, 2, 3]
```

```
y = [x**2 for x in lista]      # y = [1, 4, 9]
```

Programowanie funkcyjne - *lambda*

- Funkcja anonimowa *lambda*:

lambda argumenty: wyrażenie

- Lambda zwraca wartość wyrażenia operującego na podanych argumentach

```
lambda x: (x + 1) ** 2
```

zwraca x zwiększone o 1 i podniesione do kwadratu

Programowanie funkcyjne - narzędzia

- *Map* - stosuje podaną funkcję do zadanej sekwencji, tworząc nową sekwencję
- *Filter* – odfiltrowuje elementy sekwencji wg. Wyniku podanej funkcji
- *Reduce* – oblicza wartość z sekwencji

Funkcje

- Struktura:

```
def nazwa(argumenty) :  
    instrukcje
```

- Wartość zwracana poprzez instrukcję *return*
- Zarówno typ zwracanej wartości, jak i argumentów nie jest ustalony!
- Możliwe jest przyjmowanie/zwracanie nieokreślonej liczby argumentów/wartości

Klasy - podstawy

- Struktura:

```
class nazwa(klasy_nadrzedne):  
    atrybuty i metody
```

- W nawiasach podajemy listę klas po których nasza klasa ma dziedziczyć
- Instancję tworzymy poprzez `inst=klasa()`
- Dostęp do atrybutów/metod poprzez `klasa.atribut` lub `klasa.metoda`

Klasy - podstawy

- Konstruktor tworzymy pisząc metodę `__init__`
- Destruktor to metoda `__del__`
- Operatory przeciążamy pisząc metody o odpowiednich nazwach (np. `__getitem__`, `__add__`, `__cmp__`, ...)

Moduły - importowanie

- Moduły to pliki zawierające obiekty gotowe do użycia (zmienne, funkcje, klasy)
- Importując moduł uzyskujemy dostęp do jego zawartości
- Podczas importowania moduł jest wykonywany!

```
import modul
```

```
from modul import obiekt
```

```
reload(modul)
```

Moduły – tworzenie własnych modułów

- Modułem jest każdy plik z kodem źródłowym Pythona i rozszerzeniem *.py*
- Należy pamiętać, że podczas importowania kod modułu jest wykonywany

(...)

```
if __name__ == '__main__':  
    instrukcje
```

Pakiety i inne rozszerzenia

- Pakiet to katalog, w którym znajdują się moduły oraz specjalny plik `__init__.py` wykonywany podczas importowania pakietu
- Pakiety importujemy tak samo jak moduły
- Aplikacje w Pythonie mogą również wykorzystywać rozszerzenia napisane w C

Komentarze i dokumentacja

- Komentarze jednolinijkowe zaczynamy znakiem „#”
- Dokumentację do kodu tworzymy pisząc pod nagłówkami funkcji/klas/modułów łańcuchy znaków

```
def func():  
    """Komentarz do funkcji"""  
    pass
```

- Komentarze można odczytać np. funkcją *help*
`help(func)`
- Istnieją narzędzia automatycznie tworzące pliki dokumentacji (np. w HTML-u)

Do zgłębienia

- Zakresy: zasada LEGB, zakresy w klasach, modułach, pakietach itd.
- Projektowanie zorientowane obiektowo: enkapsulacja, dziedziczenie, ...
- Klasy *w nowym stylu*
- Wyjątki – definiowanie i wykorzystanie

Zastosowanie

Biblioteka standardowa

- Około 200 modułów
- System operacyjny, wyrażenia regularne, programowanie sieciowe, GUI, czas, bazy danych i wiele, wiele innych

Pozostałe biblioteki

- Python Package Index (PyPI) to zbiór ogromnej ilości bibliotek – stale przybywają nowe!
- Bindings w Pythonie dla niemal każdego większego projektu (Qt, GTK, MySQL, ...)
- NumPy, SciPy – obliczenia naukowe
- Django, Zope – aplikacje webowe
- ...

Kto używa Pythona

- Nokia: Qt, Symbian
- Google: GMail, Groups, Maps, YouTube
- Autodesk: Maya
- NASA
- Gry: Battlefield, Civilization, ... ;)
- Philips
- Yahoo!
- ...

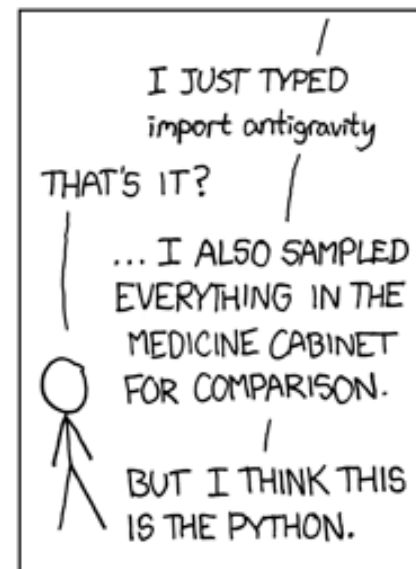
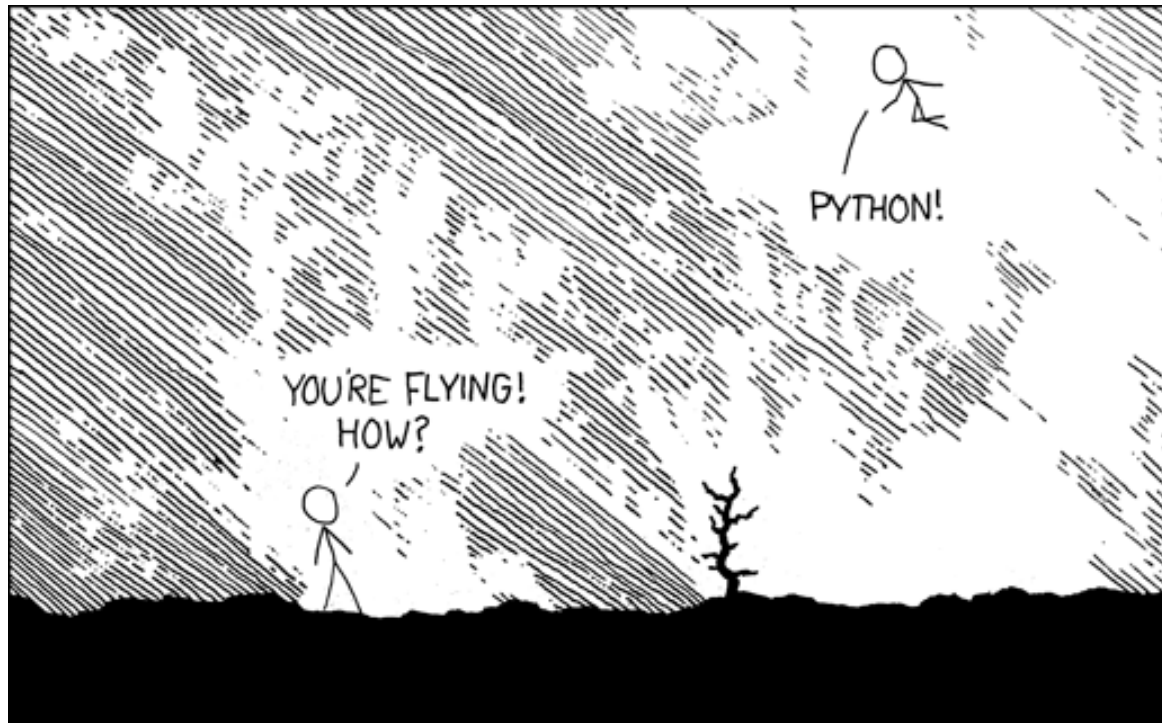
Do poczytania

- *Python. Wprowadzenie*, Mark Lutz
- *Python. Od podstaw*
- *Zanurkuj w Pythonie*, Mark Pilgrim
- dokumentacja na python.org
- Python Enhancement Proposals (PEPs)

Przykłady

Zakończenie

Programming is fun again!



Pytania?

- Śmiało! :)



Dziękuję za uwagę!

NOBODY expects the Spanish Inquisition!

