Programowanie i projektowanie obiektowe

Python

Paweł Daniluk

Wydział Fizyki

Jesień 2012

Python

- język skryptowy
- interpretowany
- typowanie dynamiczne, ale ścisłe
- egzotyczna składnia

Python

- język skryptowy
- interpretowany
- typowanie dynamiczne, ale ścisłe
- egzotyczna składnia

Doktryna

- Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Readability counts.
- To describe something as clever is NOT considered a compliment in the Python culture.
- ~~There is more than one way to do it.~~
- There should be one – and preferably only one – obvious way to do it.

Składnia

Java albo C

```
void foo(int x) {
    if (x == 0) {
        bar();
        baz();
    } else {
        qux(x);
        foo(x - 1);
    }
}
```

Python

```
def foo(x):
    if not x:
        bar()
        baz()
    else:
        qux(x)
        foo(x - 1)
```

- Wcięcia mają znaczenie semantyczne.
- Brak średników.

Instrukcje sterujące

if

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

Instrukcje sterujące c.d.

while

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
```

for

```
words = ['cat', 'window', 'defenestrate']
for w in words:
    print w, len(w)
```

Instrukcje sterujące c.d.

range

```
range(10)
range(5, 10)
range(0, 10, 3)
range(-10, -100, -30)

a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print i, a[i]
```

Listy

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
```

Listy c.d.

Stos

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

Funkcje

```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print a,
        a, b = b, a+b

fib(2000)
```

Funkcje c.d.

Domyślne wartości argumentów

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_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
```

Funkcje c.d.

Argumenty nazwane

```
def parrot(voltage, state='a stiff', action='voom',
      type='Norwegian Blue'):
    print "-- This parrot wouldn't", action,
    print "if you put", voltage, "volts through it."
    print "-- Lovely plumage, the", type
    print "-- It's", state, "!"
```

```
parrot(1000)                    # 1 positional argument
parrot(voltage=1000)            # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM') # 2 keyword args
parrot(action='VOOOOOM', voltage=1000000) # 2 keyword args
parrot('a million', 'bereft of life', 'jump')
    # 3 positional arguments
parrot('a thousand', state='pushing up the daisies')
    # 1 positional, 1 keyword
```

Generatory

```
from itertools import count

def generate_primes(stop_at=0):
    primes = []
    for n in count(2):
        if 0 < stop_at < n:
            return # raises the StopIteration exception
        composite = False
        for p in primes:
            if not n % p:
                composite = True
                break
            elif p ** 2 > n:
                break
        if not composite:
            primes.append(n)
            yield n

for i in generate_primes():  # iterate over ALL primes
    if i > 100:
        break
    print(i)
```

Klasy

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'
```

```
x = MyClass()
x.i=23456
x.f()
```

Funkcje i metody są obiektami

```
xf=x.f
xf()
```

Konstruktor

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart

x = Complex(3.0, -4.5)
```

Metody

```
class Complex:
    ...
    def add(self, c):
        self.r += c.r
        self.i += c.i
```

Statyczne

```
    @staticmethod
    def addTwo(c1, c2):
        return Complex(c1.r, c1.i).add(c2)
```

```
c1=Complex(1,2)
c2=Complex(3,4)
c3=Complex.addTwo(c1, c2)
```

Dziedziczenie

```
class Animal:
    def __init__(self, name):
        self.name=name

class Dog(Animal):
    def __init__(self, name, breed):
        Animal.__init__(self, name)
        self.breed=breed

    def talk(self):
        print "Woof!"

class Cat(Animal):
    def talk(self):
        print "Meow!"
```

Duck typing

```
class Fish(Animal):
    pass

def talk_to_me(x):
    x.talk()
```

```
>>> talk_to_me(Cat("Greebo"))
Meow!
>>> talk_to_me(Fish("Napoleon"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in talk_to_me
AttributeError: Fish instance has no attribute 'talk'
```

Expando

```
class Expando:
    pass
```

```
>>> x=Expando()
>>> x.a=2
>>> x.b=3
>>> print x.__dict__
{'a': 2, 'b': 3}
```

Dekoratory

```
def bread(func):
    def wrapper():
        print "</''''''\>"
        func()
        print "<\______/>"
    return wrapper

def ingredients(func):
    def wrapper():
        print "#tomatoes#"
        func()
        print "~salad~"
    return wrapper

def sandwich(food="--ham--"):
    print food
```

```
>>> sandwich()
--ham--
>>> sandwich = bread(ingredients(sandwich))
>>> sandwich()
</''''''\>
#tomatoes#
--ham--
```

Dekoratory c.d.

```
@bread
@ingredients
def sandwich(food="--ham--"):
    print food
```

```
>>> sandwich()
</,,,,,,\>
#tomatoes#
--ham--
~salad~
<\______/>
>>>
```