Język Boo

Język Boo

Słowem wstępu

- Odpowiednik Pytona w świecie CIL
- Automatyczne typowanie
- Statyczne typowanie
- Klauzule, generatory
- Cukier dla list, tablic przypisań, tablic

Składnia

C#

```
int x = 3;
FooBarQux fbq = make_fbq ();
char c = 'a';

expr_1 = expr_2 = expr_3;

Class c = new Class (params);

Class[] c = new Class [size]
T[] l = new T[] { expr_1, expr_1,
                  ..., expr_n }

if (cond)
  return foo;
do_something ();

if (cond) answer = 42;
```

Boo

```
x = 3
fbq = make_fbq()
c = char('a')

expr_1 = expr_2 = expr_3

c = Class(params)

c = array(Class, size)
l = (expr_1, expr_1,
        ..., expr_n)

if cond:
  return foo
do_something()

if cond:
    answer = 42
//Lub
answer = 42 if cond
```

Wyjątki

C#

```
try ...
catch (FooException e) { ... }
catch (BarException e) { ... }
```

```
try { foo (); bar (); }
catch (Exception e) { baz (); }
finally { qux (); }
```

```
throw new ArgumentException ("foo");
```

Boo

```
try:
    ...
except e as FooException:
    ...
except e as BarException:
    ...
```

```
try:
    foo()
    bar()
except e:
    baz()
ensure:
    qux()
```

```
raise ArgumentException("foo")
```

CLR

C#

```
using System;
using SWF = System.Windows.Forms;
using System.Xml;
....
Console.WriteLine ("foo");
SWF.Form x = new SWF.Form ();
XmlDocument doc = new XmlDocument ();

class Foo : Bar {
int i;
public Foo (int x) : base (x) { ... }
}

class C : I1, I2 {
void I1.m () { ... }
void I2.m () { ... }
}
```

Boo

```
import System
import System.Windows.Forms as SWF
import System.Xml

print "foo"
x = SWF.Form()
doc = XmlDocument()

class Foo (Bar):
i as int
    def constructor(x as int):
        super(x)

class C (I1, I2):
    def I1.m():
        ...
    def I2.m():
        ...
```

Najbardziej potrzebne literały

- Tablice

```
l = (,) // pusta tablica typu object
l = (1, 2, 3)
l = ("Eric", )

a = (1, 2) + (3, 4)
assert a == (1, 2, 3, 4)
array2 = array(string, mylist)
```

- Listy

```
l = [] // pusta lista
l = ["one", 2, 3]

l += ["test"]
l.Add("new item")
l.IndexOf("one")
```

- Łańcuchy znakowe, tablice i listy posiadają (o podobnej składni jak w języku Matlab) operator wycinania `[::]`

Najbardziej potrzebne literały

- Słowniki (Tablice mieszające)

```
h = {} // pusty słownik
h = { "spam" : "eggs" }

h["test"] = "nowy stary tekst"
```

- Przedziały czasu

```
year = 365d
byhourminute = ${date.Now + 1h +1m}       //$ zwraca string
System.Threading.Thread.Sleep(1s + 10ms)
```

- Wyrażenia regularne

```
fname, lname = /(\w+)/.Matches(" Eric Idle ")
print(fname)
print(lname)

//Jeżeli zawierają spacje należy poprzedzić je @
fname, lname = @/ /.Split("Eric Idle")
```

Metody jako obiekty

```
def ignore(item):
  pass

def selectAction(item as int):
  return print if item % 2
  return ignore

for i in range(10):
  selectAction(i)(i)

callable Malkovich() as Malkovich

def malkovich() as Malkovich:
  print("Malkovich!")
  return malkovich

malkovich()()()()

add = { a | return a+1 }
```

Dodatkowe metody metod

- `Invoke(argumenty metody) as <typ metody>` wywołuje funkcję, działa jak ()
- `BeginInvoke(argumenty metody, callback as AsyncCallback) as IAsyncResult` wywołuje asynchronicznie
- `EndInvoke(asyncResult as IAsyncResult) as <typ metody>` kończy wywołanie asynchroniczne, i zwraca wynik

Generatory

- Wygodna składnia definiowania zbiorów

```
even = [i*2 for i in range(5)]
students4 = [s for s in students if s.avg>4.0]
```

- Ale to nie wszystkie liczby parzyste a studentów jest wielu.
- Na szczęście zbiory te są proste i można wyrazić je bezpośrednio

```
allevens = a for a in inf() if a%2 == 0
students4 = s for s as student in students if s.avg>=4.0
```

- Implementują `IEnumerable`, czyli:

```
for i in students4:
  print i
```

- Czy `pa = [a for a in parzyste if a < 10]` jest poprawne ?

Metody Generujące

- Potrafią zwracać wyniki wielokrotnie dzięki kluczowemu słowu `yield`

```
def inf():
  a = 0
  while true:
    yield a
    a++
```

- Metoda pamięta swoje zmienne lokalne i dzięki temu jest wydajna i łatwa do zrozumienia.
- Sposobem na ograniczenie ilości generowanych danych jest wbudowana konstrukcja `zip`:

```
set10 = [x for i,x in zip(range(10), inf())]
```

Generatory a sprawa praktyczna

- Generatory tworzą dane w chwili kiedy są potrzebne
- Czy mają jednak praktyczne zastosowanie ?
- Pozwalają dość ładnie wyrazić zapytania SQL, czy XML

```
def selectElements(element as XmlElement, tagName as string):
  for node as XmlNode in element.ChildNodes:
    if node isa XmlElement and tagName == node.Name:
      yield node
```

Duck typing

- Jeżeli chodzi jak kaczka i kwaka jak kaczka to jest to kaczka
- Czasami warto zrezygnować ze statycznego typowania

```
s as int  //to jest liczba
d as duck //a to cokolwiek

d = "ala"
s = 1;
d = s + 1
s = d
```

- Jest to tez zaawansowany cukier składniowy dla `Invoke` przydatny m.in. w technologii COM