Wstęp do programowania

Podstawowe konstrukcje programistyczne

Paweł Daniluk

Wydział Fizyki

Jesień 2013

Przypomnienie

Programowanie imperatywne

- Program składa się z instrukcji, które są wykonywane po kolei.
- Instrukcje mogą zmieniać stan programu.

Stan programu

Sprzętowo: zawartość pamięci i rejestrów procesora.
Matematycznie: Wartościowanie zmiennych.

Podstawowe konstrukcje

Komentarze

```
"""This is a multi-line comment.
It may occupy more than one line."""
```

```
# This is an end-of-line comment
```

Podstawowe konstrukcje c.d.

Zmienne i przypisania

```
i = 5
```

Uwaga

= oznacza przypisanie

== jest operatorem porównania

Operatory arytmetyczne

Operatory porównania

==	$=$
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$
!=	$\neq$

Operatory arytmetyczne

+	$+$
-	$-$
*	$\cdot$
/	$\div$
//	dzielenie całkowitoliczbowe
%	mod
**	potęgowanie

Przykład

Podnoszenie do kwadratu

```
x = 5
y = x * x
```

Wypisywanie wartości

```
print y
```

Instrukcje sterujące

Instrukcje sterujące służą do zmiany kolejności wykonywania instrukcji w programie.
Bez nich każda instrukcja zostałaby wykonana dokładnie raz, w kolejności występowania w kodzie źródłowym.

Instrukcje warunkowe

Instrukcja warunkowa

```
if i==3:
    doSomething()
```

```
if i==3:
    doSomething()
else:
    doSomethingElse()
```

```
if i==3:
    doSomething()
elif i==2:
    doSomethingElse()
else:
    doSomethingDifferent()
```

Dygresja o wcięciach

W Pythonie nie ma nawiasów klamrowych {, } jak w C lub Javie, ani słów kluczowych `begin`, `end`. O granicach bloków kodu decydują wcięcia.

To są różne programy

```
if i==3:
    x=2
    y=3
```

```
if i==3:
    x=2
y=3
```

A ten program jest niepoprawny

```
if i==3:
    x=2
y=4
else:
    z=5
```

Pętle

while

```
while i<5:
    doSomething()
```

```
while i<5:
    doSomething()
else:
    doSomethingElse()
```

Pętle c.d.

for

```
for i in list:
    doSomething(i)
```

for

```
for i in list:
    doSomething(i)
else:
    doSomethingElse()
```

W Pythonie nie ma tradycyjnej pętli for

Java, C

```
for (i = 0; i < 10; i++) {
    doSomething(i);
}
```

Pascal

```
for i:=0 to 9 do
  begin
    doSomething(i);
  end
```

Ale jest funkcja range

```
range(3)==[0,1,2]
```

Python

```
for i in range[9]:
    doSomething(i)
```

pass

Instrukcja pusta

```
if a==2:
    pass
```

break

Opuszcza aktualnie wykonywaną pętlę.

```
while True:
    isDone=doSomething()

    if isDone:
        break
```

continue

Przechodzi do następnej iteracji pętli.

```
while i<10:
    skip=check(i)

    if skip:
        continue

    doSomething()
```

Tablice?

Short answer

Nie ma tablic.

Tablice?

Short answer

Nie ma tablic.

Są listy.

Listy

Składnia

```
[el1 , el2 , el3 , . . . ]
```

Przykład

```
a = ['spam', 'eggs', 100, 1234]
a[2] = a[3] - 233
```

Elementy są numerowane od 0.

Wycinki (ang. *slices*)

Ujemne indeksy

`a[-1]` – oznacza ostatni element listy
`a[-i]` – oznacza i-ty element od końca (miły fakt $-i \equiv_N N - i$)

Wycinki

`a[i:j]` – oznacza listę zawierającą elementy od i-tego do $j-1$-szego włącznie
i i j mogą być ujemne. Można je również pominąć.

Wycinki z krokiem

`a[i:j:k]` – j.w. ale co k-ty element

Funkcje

Funkcje w matematyce

$$f : D \longrightarrow W$$

D – dziedzina
W – zbiór wartości

Funkcja może być wieloargumentowa

$$f : D_1 \times D_2 \times \cdots \times D_n \longrightarrow W$$

Funkcje c.d.

Funkcje w Pythonie

```
def f(arg1, arg2, ..., argN):
    compute
    compute
    ...
    compute

    return result
```

Programowanie imperatywne!

Obliczenia w funkcji mogą zależeć i zmieniać stan programu. Zatem funkcja wywołana wielokrotnie dla tych samych argumentów może dawać różne wyniki.

Funkcje c.d.

Przykład

```
n=0
def f():
    global n
    n=n+1
    return n

print f()
print f()
```

Python domyślnie traktuje wszystkie zmienne występujące w funkcji jako lokalne, aby uniknąć takich sytuacji. Słowo kluczowe `global` powoduje odwołanie do zmiennej `n` występującej na zewnątrz funkcji.

Funkcje c.d.

Użyteczny przykład

```
def fib(n): # return the nth Fibonacci number
    a, b = 0, 1
    i = 1
    while a < n:
        a, b = b, a+b
        i = i + 1
    return b
```

Interpreter Pythona

Praca interaktywna

```
pawel@tok ~> python
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> i=3
>>> print i
3
>>> a=range(5)
>>> a
[0, 1, 2, 3, 4]
>>> i=0
>>> while i<5:
...     print i
...     i=i+1
...
0
1
2
3
4
>>>
```

Interpreter Pythona

Skrypt wykonywalny

```
#!/usr/bin/python

print "Hello, World!"
```

Dobrze jest pamiętać o `chmod a+x ...`.

W trybie interaktywnym

```
>>> execfile("hello.py")
Hello, World!
>>>
```

Zadanie 1

Napisz programik, który:

1. Wczytuje dwie liczby i oblicza ich średnią arytmetyczną.
2. Wczytuje liczbę i sprawdza, czy jest dodatnia - algorytm powinien zwrócić odpowiedź TAK albo NIE.
3. Wczytuje dwie liczby i wyznacza najmniejszą z nich.
4. Wczytuje trzy liczby i wyznacza największą z nich.
5. Wczytuje dwie liczby a, n $(n \in N)$ i oblicza a^n.
6. Wczytuje dwie liczby a, n $(n \in Z)$ i oblicza a^n.

Wczytywanie

```
>>> a=input("Podaj liczbe:")
Podaj liczbe:1
>>> a
1
>>> (a,b)=input("Podaj dwie liczby:")
Podaj dwie liczby:2,3
>>> a
2
>>> b
3
```

Algorytm Euklidesa

Pomysł

$$NWD(a,b) = \begin{cases} a & \text{jeśli } b = 0 \\ NWD(b,a) & \text{jeśli } b > a \\ NWD(a-b,b) & \text{jeśli } a \geq b > 0 \end{cases}$$

Naiwna implementacja

Pseudokod

NWD(a, b)

▷ zakładamy, że $a > b$ i $a + b > 0$

1 **while** $b > 0$
2 **do** $a \leftarrow a - b$
3 **if** $b > a$
4 **then** swap(a, b)
5 **return** a

Niska wydajność

Dla $a = 10^{10}$ i $b = 1$ trzeba wykonać 10^{10} odejmowań.

Poprawna implementacja

Pseudokod

NWD(a, b)

▷ zakładamy, że $a > b$ i $a + b > 0$

1 **while** $b > 0$

2 **do** $a \leftarrow a \mod b$

3 swap(a, b)

4 **return** a

Zadanie 2

Zaimplementuj algorytm Euklidesa.

Zadanie 3

Stwórz listę długości 10 zawierającą:

1. Kolejne liczby naturalne.
2. Kolejne liczby nieparzyste.
3. Kolejne sumy częściowe szeregu liczb naturalnych.
4. Kolejne liczby Fibonacciego.
5. Losowe wartości ze zbioru $\{0, 1\}$.

Zadanie 3

Stwórz listę długości 10 zawierającą:

1. Kolejne liczby naturalne.
2. Kolejne liczby nieparzyste.
3. Kolejne sumy częściowe szeregu liczb naturalnych.
4. Kolejne liczby Fibonacciego.
5. Losowe wartości ze zbioru $\{0, 1\}$.

Generator liczb pseudolosowych

```
>> import random
>>> random.randint(0,1)
0
>>> random.randint(0,1)
1
>>> random.randint(0,1)
0
```

Zadanie 4 – Flaga polska

Zadanie

Lista a wypełniona zerami i jedynkami reprezentuje ciąg n urn w których znajdują się żetony białe (0) i czerwone (1). Podać algorytm, który zamieniając żetony miejscami doprowadzi do sytuacji, w której wszystkie żetony białe znajdują się na lewo od czerwonych.

Zadanie 4 – Flaga polska

Zadanie

Lista a wypełniona zerami i jedynkami reprezentuje ciąg n urn w których znajdują się żetony białe (0) i czerwone (1). Podać algorytm, który zamieniając żetony miejscami doprowadzi do sytuacji, w której wszystkie żetony białe znajdują się na lewo od czerwonych.

Wskazówka

Należy przesuwać się indeksem c od początku tablicy, zaś indeksem b od końca. Intencją jest utrzymywanie następującego niezmmiennika: wszystkie elementy tablicy o indeksach mniejszych od c są czerwone, zaś wiekszych od b są białe. Indeksy c i b będą się do siebie zbliżać i ostatecznie gdy c będzie równe b, to tablica będzie uporządkowana.

Strona wykładu

http://bioexploratorium.pl/wiki/Wstęp_do_programowania_2013z