Systemy operacyjne 13

Spis treści

- 1 Obsługa interpretera, wykonywanie skryptów
  - 1.1 Praca interaktywna
  - 1.2 Pliki skryptowe
- 2 Zmienne
  - 2.1 Zmienne proste
  - 2.2 Zmienne tablicowe
  - 2.3 Zmienne strukturalne
- 3 Interakcja z użytkownikiem
  - 3.1 Wyświetlanie danych
  - 3.2 Pobieranie danych
  - 3.3 Obsługa plików
- 4 Operacje logiczne i arytmetyczne
  - 4.1 Operacje arytmetyczne
  - 4.2 Operacje logiczne
- 5 Wybrane struktury języka
  - 5.1 Warunek if
  - 5.2 Pętla while
  - 5.3 Pętla for
- 6 Procedury, funkcje, obiekty
  - 6.1 Funkcje
- 7 Co dalej?
- 8 Zadania

Obsługa interpretera, wykonywanie skryptów

Pythona można używać zarówno jak normalny interpreter - "zlecać" mu wykonanie programów zapisanych w plikach źródłowych lub w trybie interaktywnym.

Praca interaktywna

W trybie interaktywnym Python pobiera os użytkownika kolejne polecenia i na bieżąco je interpretuje. Aby uruchomić Pythona w tym trybie wystarczy wpisać:

```
python
```

Python będzie wówczas interpretował każde polecenie. Np.:

```
2+2
4
```

Aby opuścić środowisko należy posłużyć się kombinacją Ctrl-D.

Pliki skryptowe

Aby plik a kodem został wykonany należy podać go jako parametr interpretera:

```
python plik.py
```

lub dodać linię:

```
#!/usr/bin/python
```

na początku skryptu i ustawić plikowi prawo wykonania. Aby móc używać polskich liter w można dodać w drugiej linii:

```
# -*- coding: iso-8859-2 -*-
```

Zmienne

Zmienne proste

Sposób w jaki Python obsługuje zmienne jest jedną z jego głównych zalet. Nie istnieje potrzeba deklarowania zmiennych. Ich typ określany jest podczas przypisania. Mimo to Python, w przeciwieństwie do wielu innych języków skryptowych kontroluje typy zmiennych podczas wykonywania operacji. Poniżej pokazano przykładowe przypisania:

```
cztery = 2 + 2
x = y = z = 0
a, b = b, a+b
```

Zmienne tablicowe

Przed ostatni przykład pokazuje prostotę deklarowania tablic, zaś ostatni możliwość przypisania kilku zmiennych na raz (czy jak kto woli "rozpakowywanie tablic"). Bardzo cenną własnością Pythona jest prostota obsługi tablic. Utworzenie tablicy będzie miało postać:

```
pierwsze = [2, 3, 5, 7]
```

Użycie takiej zmiennej jest równie proste:

```
print pierwsze
[2, 3, 5, 7]
print pierwsze[2]
5
print pierwsze[1:3]
[3, 5]
```

Ostatni przykład pokazuje sposób operowania na fragmencie tablicy.

Zmienne strukturalne

Z punktu widzenia języka Python tablice są szczególnym przypadkiem bardziej ogólnej struktury - listy. Tablice charakteryzuje jeden typ danych określający wszystkie jej elementy. W Pythonie istnieje możliwość przypisywania poszczególnym elementom wartości dowolnego typu:

```
lista = [1, 3.4, 'tekst', 8]
```

Poza omówionym wcześnie sposobem obsługi list istnieje szereg metod pozwalających używać list jako najważniejszych, prostych struktur danych:

len(a)
: Podaje rozmiar listy.

append(x)
: Dodaje element do kończ listy, odpowiednik a[len(a):] = [x].

extend(L)
: Rozszerza listę poprzez dołączenie wszystkich elementów podanej listy L, odpowiednik a[len(a):] = L.

insert(i, x)
: Wstawia element na podaną pozycję listy. Pierwszym argumentem wywołania jest indeks elementu, przed którym nowy element ma zostać wstawiony: tak więc a.insert(0,x) wstawia na początek listy, a a.insert(len(a),x) jest odpowiednikiem a.append(x)

remove(x)
: Usuwa pierwszy napotkany element z listy, którego wartością jest x. Jeżeli nie ma na liście takiego elementu, zgłaszany jest błąd.

pop([i])
: Usuwa element z podanej pozycji na liście i zwraca go jako wynik. Jeżeli nie podano żadnego indeksu a.pop(), zwraca ostatni element na liście. Oczywiście, jest on z niej usuwany.

index(x)
: Zwraca indeks pierwszego elementu listy, którego wartością jest x. Jeżeli nie ma takiego elementu zgłaszany jest błąd.

count(x)
: Zwraca liczbę wystąpień elementu x na liście.

sort()
: Sortuje elementy na liście, w niej samej. (W wyniku nie jest tworzona nowa, posortowana lista)

reverse()
: Odwraca porządek elementów listy, również w niej samej.

Przykład:

```
>>> a = [66.6, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.6), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.6, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.6, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.6]
>>> a.sort()
>>> a
[-1, 1, 66.6, 333, 333, 1234.5]
```

Do konwersji ciągu znaków oddzielonych określonym separatorem na tablicę, często przydaje się metoda 'split'. Wywołuje się ja na rzecz zmiennej typu tekstowego. Pierwszy jej parametr określa separator. Przykład:

```
>>> a='123,qwe,asd,zxc'
>>> b = a.split(',')
>>> b
['123', 'qwe', 'asd', 'zxc']
>>>
```

Interakcja z użytkownikiem

Wyświetlanie danych

Do wyświetlania komunikatów w języku Python służy polecenie *print*. Zatem klasyczny "pierwszy program" będzie miał postać:

```
print 'Hello world'
```

Oczywiście parametrem *print* mogą być również zmienne. Kolejne parametry oddziela się przecinkami:

```
print a, ' do kwadratu wynosi ', a*a
```

Pobieranie danych

Do wprowadzania danych służą funkcje *input* i *raw_input*. Pierwsza z nich pobiera dane i interpretuje je zgodnie ze składnią Pythona. Czyli gdy chcemy uzyskać wartość typu *string* to wprowadzany tekst musi zostać ujęty w cudzysłów. *input* akceptuje także nazwy zmiennych. Przykład:

```
a = input("Podaj swoj wiek: ")
```

Parametr funkcji *input* jest tekstem wyświetlanym przed wprowadzeniem danych przez użytkownika. Powinno to być jakieś pytanie lub znak zachęty. Jeśli chcemy aby wprowadzane teksty były zawsze zapisywane jako zmienna typu *string* należy użyć polecenie *raw_input*:

```
a = input("Podaj swoje imie: ")
```

Obsługa plików

Aby otworzyć plik należy wydać polecenie:

```
uchwyt = open(nazw_pliku, tryb)
```

Gdzie *uchwyt* to zmienna identyfikująca plik, *nazwa_pliku* to ścieżka dostępu do pliku, zaś *tryb* to jedna z następujących wartości:

- 'r' - plik otwarty tylko do odczytu
- 'w' - plik otwarty tylko do zapisu
- 'a' - plik otwarty do dopisywanie nowych danych na końcu
- 'r+' - plik otwarty do odczytu i zapisu

Na przykład:

```
f = open('/etc/passwd', 'r')
```

Aby odczytać wartość z pliku można posłużyć się metodą *read*. Na przykład:

```
a = f.read(10)
```

Wartość 10 oznacza w tym przypadku maksymalna liczbę znaków, które zostaną odczytane. Jeśli chcemy przeczytać cały plik, możemy parametr pominąć.

Istnieje także możliwość odczytania jednej linii z pliku. Służy do tego metoda *readlines*. Na przykład:

```
linia = f.readlines()
```

Zapis do pliku odbywa się analogicznie, metodą *write*. Parametrem tej metody jest tekst, który ma do pliku trafić. Na przykład:

```
f.write('Tekst do zapisania')
```

Na koniec warto zamknąć plik:

```
f.close
```

Operacje logiczne i arytmetyczne

Operacje arytmetyczne

W Pythonie wbudowano wszystkie istotne operatory matematyczne. Ich obsługa jest :

- +, -, *, /
- dzielenie całkowite: //
- reszta z dzielenie (modulo): %
- potęgowanie: **

Operację wykonywane są w zróżnicowany sposób w zależności od typów operandów. Domyślnie wynik jest tego samego typu co operandy. Czyli dzielenie wykonane na liczbach całkowitych zwróci wartość całkowitą. Jeżeli jeden z operandów jest rzeczywisty dzielenie zwróci wartość rzeczywistą. Poniższy przykład obrazuje ten mechanizm:

```
(1*4) / 3
1

(1.0*4) / 3
1.33333333333333

(1.0*4) // 3
1.0
```

Operacje logiczne

W wyrażeniach logicznych można użyć zmiennych dowolnego typu. Mają one następujące znaczenie logiczne:

- Fałsz: 0, False, None, [], {}
- Prawda: True, wartości niepuste

Dodatkowo zaimplementowane są # -*- coding: iso-8859-2 -*-operatory logiczne:

- Spójniki: and, or, not
- Operatory: ==, !=, 1 < x < 2

Wybrane struktury języka

Python posiada wszystkie istotne struktury językowe jak: *if*, *while*, *for*. Cechą charakterystyczną jest sposób oznaczania instrukcji złożonej, czyli ciągu instrukcji znajdujących się "wewnątrz" struktury. Python nie posiada specjalnych słów kluczowych określających takie bloki (jak w pascalu: begin i end

oraz nawiasy klamrowe w C). Poszczególne bloki oznaczone są **głębokością wcięcia**. Przy czym sama głębokość nie ma znaczenia a istotnym jest jedynie aby instrukcje znajdujące się w jednym bloku były wcięte tak samo:

```
blok1
blok1
  blok2
  blok2
    blok3
blok1
```

Warunek if

Podstawą programowania jest instrukcja warunkowa. Ma ona następującą składnię

```
if warunek:
  instrukcja
```

Jej zastosowanie prezentuje poniższy przykład:

```
x = 4
if x % 2 == 0:
  print "parzysta"
  print "x = ", 2
```

Pętla while

Najprostszą pętlą w pythonie jest pętla *while*. Składnia tej struktury jest następująca:

```
while warunek_kontynuacji:
  instrukcja
```

Pętla będzie wykonywała się tak długo póki warunek pozostanie spełniony. W przypadku gdy przyjemnie wartość *False* pętla zakończy swoje działanie. Oto przykład zastosowania tej struktury:

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

Pętla for

Koleją pętlą jest struktura *for*. Składnia tej instrukcji jest następująca:

```
for zmienna in tablica:
  instrukcja
```

Wykonuje się ona tyle razy ile elementów znajduje się w tablicy po słowie *in*. Zmienna przyjmuje kolejne wartości z tablicy. Prezentuje to przykład:

```
a = [1,2,3,4]
for e in a:
  print e
  print "koniec"
```

Aby użyć typowej funkcjonalności pętli *for*, czyli wykonywania dla kolejnych wartości całkowitych, należy wspomóc się funkcją *range*. Ma ona następującą składnię:

```
range (stop)
range (start,stop)
range (start,stop,step)
```

start
: wartość początkowa (domyślnie 0)

stop
: wartość końcowa (jednak należy pamiętać, że jest do przedział otwarty, więc jeżeli wpiszemy 10 to ostatnią liczbą w przedziale będzie 9)

step
: jest to odstęp pomiędzy kolejnymi wartościami

Pętla oparta na *range* będzie może mieć postać:

```
suma = 0
for i in range(100):
  suma = suma + i
  print "suma=" , suma
```

W celu optymalizacji kodu python daje możliwość deklarowania funkcji, które mogą być wykorzystywane wielokrotnie w programie. Aby zadeklarować funkcję należy zastosować słowo kluczowe *def* zgodnie ze składnią:

Procedury, funkcje, obiekty

Funkcje

```
def nazwa_funkcji(parametr1=wartosc_domyslana, parametr2=wartosc_domyslna, ...)
  instrukcja
  return wartosc_zwracana
```

Wartości domyślne parametrów zostają użyte w przypadku, gdy wywołano funkcję z mniejszą liczbą parametrów niż zadeklarowano. Jeśli parametr nie ma wartości domyślnej musi zostać podany podczas wywołania. Instrukcja *return* jest opcjonalna. Przykład deklaracji podano poniżej:

```
def funkcja(arg1, arg2=1, arg3=[]):
  print arg1, arg2, arg3
  return 4
```

Poniżej pokazano przykłady wywołania zadeklarowanej wcześniej funkcji funkcji:

```
funkcja("jeden", [3])
funkcja(arg1="dwa", arg3=3)
print funkcja(1, 2, 3)
```

Co dalej?

Możliwości składni Pythona są dużo większe od przedstawionych w tym punkcie. Python jest zaawansowanym językiem obiektowym, pozwalającym, poprzez wykorzystanie zewnętrznych bibliotek, na pisanie programów wyposażonych w GUI a nawet opartych na OpenGLu. Zewnętrzne biblioteki pozwalają na pisanie między innymi aplikacji sieciowych, bazodanowych, multimedialnych... Osoby, które zainteresowałem programowanie w Pythonie odsyłam do szczegółowej dokumentacji znajdującej się w internecie http://www.python.org.pl/ lub do samouczków ("tutoriali") np.: http://www.python.org.pl/tut/tut.html

Zadania

- Napisz skrypt zapisujący informacje o systemie do pliku tekstowego.
- Napisz skrypt pobierający liczby z pliku i wyświetlający ich sumę.
- Z pliku */etc/passwd* wyświetl w dwóch kolumnach nazwy użytkowników i ich katalogi domowe.
- Napisz program oparty na pętli for wypisujący następujący "tekst":

```
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
```

- Zmodyfikuj pogram aby wyświetlał:

```
    o
   oo
  ooo
 oooo
ooooo
```

- Napisz program wykonujący jedno z czterech działań na dwóch liczbach. Rozbuduj go w taki sposób aby po wykonaniu zadania program prosił o podanie nowego działania. Program ma kończyć działanie po wpisaniu przez użytkownika słowa „koniec”.
- Napisz program wyświetlający 30 elementów ciągu Fibonacciego.
- Napisz program wyświetlający liczby pierwsze mniejsze od 100. Czy stosując tablice można przyspieszyć działanie programu?