ZAJĘCIA NR 3
I DALSZE

Zawartość

Autor: ŁUKASZ CZERWIŃSKI

L.Czerwinski@students.mimuw.edu.pl
CzerwinskiLukasz1@gmail.com

2012-10-19

--- OPERATORY LOGICZNE (AND, OR, NOT)

http://docs.python.org/library/stdtypes.html#boolean-operations-and-or-not

Formalna definicja (**ważne!**)

x or y	if x is false, then y, else x	(1)
x and y	if x is false, then x, else y	(2)
not x	if x is false, then True, else False	(3)

UWAGA!

```
"a" and "b" # daje "b" (a nie True!)
"a" or "b" # daje "a" (a nie True!)
```

ale:

```
not "a" daje False # no bo co innego? ;)
```

➔ *Priorytet operatorów [i kilka krótkich zadanek-łamigłówek logicznych]*

Czemu ważne jest określenie priorytetów?

Odpowiedź: Bo wyrażenie logiczne w zależności od priorytetów (nawiasowania) może dać nam różne wyniki.

*** R Zadanie 1. Priorytety or i and.

Skonstruuj przykład (*de facto* 2 przykłady – patrz rozwiązanie poniżej) złożony wyłącznie z `False, True, and` i `or`, który ustali priorytet między `or` i `and` (tzn. wynik `False` oznacza jedną możliwość, a `True` – drugą).

** R Zadanie 2. Ustal priorytety or i and.

Określ priorytet `and`/`or` na podstawie wyników powyższych przykładów.

*** R Zadanie 3. Ustal priorytety not.

Skonstruuj przykład lub przykłady ustalające priorytet dla not względem pozostałych (podpowiedź: osobno sprawdzić z andem i osobno z orem)

➔ *Rozwiązania zadań*

Zadanie 1:

```
False and True or True
```

Wyjaśnienie:

`False` – oznacza, że `or` jest ważniejszy, `True` – że `and` (przy założeniu, że liczy się tylko priorytet, a nie sama kolejność występowania)

Istotność kolejności występowania można łatwo sprawdzić – przy użyciu lustrzanego przykładu:

```
True or True and False
```

Jeśli drugie wyrażenie da inny wynik, niż pierwsze – kolejność działań ma znaczenie [ale nie daje, więc nie ma]

Zadanie 2:

```
False and True or True  # wynik to True
True or True and False # wynik to również True
```

W takim razie `and` jest ważniejszy.

Zadanie 3:

```
not True or True # True
not False and False # False
```

Czyli priorytet jest taki:

x or x and not y ⇔ x or (x and (not y))

--- FUNKCJE WBUDOWANE – WYWOŁYWANIE, IMPORT MODUŁU

Terminy: argumenty, wywołanie funkcji, wartość zwracana, funkcje wbudowane, funkcje własne, moduł, import modułu.

 # len (przypomnienie)

Funkcji len używaliśmy, nie wiedząc jeszcze, co to jest funkcja. Teraz użyjmy jej świadomie:

```
print len("Jakiś napis") # 11
```

 # abs

```
print abs(4)
print abs(-4)
```

 # str

```
print 4
print str(4)

print "4" + "4"
a = "4"
print a + "4" # działa, bo a jest stringiem

b = 4
print b + "4" # nie działa
print str(b) + "4" # działa
```

 # funkcje można zagnieżdżać

```
print abs(-4) + "4" # tak nie działa
print str(abs(-4)) + "4" # tak działa
```

 # zmienna liczba argumentów

```
print str(4)
print str()
```

Sprawdź, co wyświetli:

```
print abs()
```

 # import modułu (5-10 min.)

```
import math
print math.cos(0)

from math import * # lub from math import cos, sin
print cos(0)
```

--- WŁASNE FUNKCJE

Do czego przydają się funkcje?

Jeśli wykonujemy jakąś bardziej skomplikowaną czynność kilka razy w różnych miejscach programu, zachodzi pokusa skopiowania kodu kilka razy. Niestety później, w razie błędów lub zmieniania działania tego kodu, trzeba być bardzo czujnym, by pamiętać o poprawieniu wszystkich miejsc, gdzie ten kod wystąpił. W praktyce bardzo często programista o którymś miejscu zapomni albo go nie zauważy – prawdziwe programy potrafią liczyć wiele tysięcy linii kodu!

Skoro program potrafią być duże, to wyobraź sobie, jak wygląda kod pisany „ciurkiem" linijka po linijce, przez tysiące linii. Koszmar! Nawet jeśli podzielimy go na kawałki odstępami i komentarzami, łatwo można się pogubić, bo mniej ważne fragmenty kodu są razem z ważniejszymi, a kod pomocniczy przeplata się z główną akcją programu.

Na oba powyższe problemy pomaga często zastosowanie funkcji, czyli wydzielenie pewnych operacji w oddzielne miejsce i nazwanie tej operacji. W naszych przykładach funkcje będą miały zazwyczaj po kilka linijek z printami i ifami, jednak w prawdziwych programach często funkcje wywołują inne funkcje itd.
Takie pisanie kodu pomaga wydzielić moduły, części kodu, które razem tworzą całość, działają w tym samym celu.

Poniżej przykłady definiowania i używania funkcji:

```
def myprint(x):
  print "Wypisuje " + x

myprint("ala ma kota")
myprint(234) # zanim sprawdzisz, zastanów się, jaki będzie wynik :)
myprint([1, 3, a]) # zanim sprawdzisz, zastanów się, jaki będzie wynik :)
```

Pytanie „na marginesie": czy powyższa funkcja zawsze działa?

Odpowiedź:
Nie zawsze – dla liczb nie zadziała (wyświetli błąd). Pamiętasz, dlaczego? Popraw to.

Poniżej poprawne rozwiązanie, ale nie patrz na nie zbyt szybko! Spójrz na nie dopiero, gdy wymyślisz poprawkę.

```
def myprint(x):
  print "Wypisuje " + str(x)

myprint("ala ma kota")
myprint(234) # a czy teraz zadziała?
myprint([1, 3, a]) # a czy teraz zadziała?
myprint([1, 3, "a"]) # a czy teraz zadziała?
```

 # z argumentem domyślnym

```
def myprint2(x, nl=True):
  print "Wypisuje " + str(x),
  if nl:
    print

myprint2(234, False)
myprint2([], True)
myprint2([1, 3, "a"])
```

 * Zadanie 1. Podwojony cosinus.

Napisz funkcję, która wypisze podwojony cosinus z liczby podanej w parametrze.

 * Zadanie 2. Pitagoras.

Napisz funkcję `pitagoras(a, b)`, która dla danych boków a i b trójkąta prostokątnego zwróci długość trzeciego boku.

 ** Zadanie 3. Przerób na funkcje.

Zastanów się, które zadania z poprzednich działów można by uprościć, korzystając z funkcji. Wybierz 2-3 z nich i uprość je.

 ** Zadanie 4. Fibonacci.

Napisz funkcję `fibonacci(n)`, która wypisuje n-tą liczbę Fibonacciego. Napisz program, który, korzystając z tej funkcji, wypisze trzecią, piątą i trzynastą liczbę Fibonacciego.

 ** Zadanie 5. Powtórz napis.

Napisz funkcję `powtórz(napis, n)`, która zwróci tekst <*napis*> powtórzony <*n*> razy.

 ** Zadanie 6. Ulepszony kalkulator.

Napisz kalkulator, korzystając z funkcji.
Propozycja: napisz funkcję, która wczyta dwie liczby i działanie, a następnie zwróci je jako trzyelementową listę.

 ** Zadanie 7. Wyśrodkowanie napisu.

Napisz funkcję `wyśrodkuj(napis, n)`, która, zakładając, że szerokość całej linii wynosi <*n*> znaków stałej szerokości, zwróci tekst, w którym <*napis*> będzie wyśrodkowany przy pomocy spacji.
Wypisz przy pomocy Twojej funkcji kilka napisów i sprawdź czy na pewno dobrze działa (sprawdź szczególnie jej działanie przy niewielkich wartościach <*n*>).

 ** Zadanie 7b. Argument domyślny.

Zmodyfikuj funkcję z poprzedniego zadania tak, by jako trzeci **opcjonalny** parametr przyjmowała znak, który ma zostać zastosowany zamiast spacji. Jako wartość domyślną przyjmij spację.

 ** Zadanie 8. Jaki trójkąt?

Napisz funkcję `jaki_trojkat(a, b, c)`, która dla danych trzech boków trójkąta a, b i c zwróci: „prostokątny”, „ostrokątny” lub „rozwartokątny”.
Bonus za skorzystanie w tym zadaniu z funkcji `pitagoras(a, b)`.

--- OBIEKTY

Terminy: obiekt (kontener na zmienne i funkcje), konstruktor (funkcja inicjująca), pole (zmienna w obiekcie), metoda (funkcja w obiekcie).

Co to jest obiekt?

Jest to „twór” istniejący w programach, który grupuje pola (czyli zmienne – stan obiektu) oraz metody (czyli funkcje, które mogą być na tych zmiennych wykonane).

Dlaczego obiekty się przydają?

Weźmy taki przykład: załóżmy, że najpierw piszemy prostą funkcję, która potrafi wyświetlać kolorowe napisy.
Teraz chcemy teraz napisać program, który będzie wyświetlać napisy na zmianę na dwa kolory – np. raz na czerwono, raz na zielono. Jak możemy to zrobić?

Rozwiązanie 1:

Napisać dwie funkcje – jedna będzie wyświetlać napisy czerwone, a druga – zielone.

Problem:

Co jeśli będziemy chcieli dodać jeszcze kilka innych kolorów? Dla każdego będziemy musieli tworzyć kolejną funkcję, prawdopodobnie kopiując i wklejając ten sam kod, i nieznacznie go modyfikując.

A co, jeśli byśmy chcieli, żeby użytkownik mógł podać dowolny kolor? Trzeba napisać całkiem inną funkcję – taką, która przyjmuje jako drugi argument kolor. W takim razie może od razu ją napiszmy… i tu dochodzimy do rozwiązania drugiego.

Rozwiązanie 2:

Stwórzmy funkcję dwuargumentową, która przyjmuje tekst oraz kolor i przy jej pomocy wyświetlajmy napisy. Zauważmy, że musimy teraz kopiować i nazwę funkcji, i jej argument (kolor) – albo wpisywać ręcznie, uważając na literówki.

A co, jeśli zechcemy nie tylko wyświetlać napisy na dwa kolory, ale i ustawić np. dwa różne kolory tła, inną pozycję tekstu (wcięcie spacjami) dla napisów czerwonych, a inną dla zielonych, albo inną wielkość liter? Robi się wiele argumentów, czyli misz-masz w zmiennych – a wszystkie musimy cały czas kopiować.

Rozwiązanie 3:

Pomysł jest następujący: stwórzmy dwie "zamknięte kapsułki", z których każda przechowuje "stan" (kolor + kolor tła + wielkość liter itp.), który ustawiamy na samym początku. Jednej z „kapsułek” przypiszmy czerwony kolor napisów, a drugiej – zielony. Następnie na przemian wywołujmy operację wypisz (z odpowiednim napisem w argumencie) raz na jednej, raz na drugiej „kapsułce”. Ten sposób jest wygodniejszy – nie musimy podawać w argumentach wywołań ani kolorów, ani żadnych innych parametrów formatowania tekstu – o wszystko dba „kapsułka”. Tą właśnie kapsułką jest OBIEKT.

Czy wyobrażasz sobie, jak mogłaby wyglądać funkcja, o której mowa w rozwiązaniu 1? Załóż, że wyświetlenie koloru polega na wypisaniu wartości pewnej zmiennej przed napisem, a wartości innej zmiennej po napisie. Spróbuj ją napisać!

Przy wypisywaniu napisów skorzystamy z biblioteki colorama. Dostaniecie ją od prowadzącego, możecie ją też ściągnąć ze strony http://pypi.python.org/pypi/colorama.

Poniżej kod funkcji z rozwiązania 1:

```
from lib_colorama import Fore, Back, Style

def wyswietl(tekst, kolor):
  print kolor + tekst + Fore.RESET

wyswietl("To jest pierwszy kolor", Fore.RED)
wyswietl("To jest drugi kolor", Fore.GREEN)
wyswietl("To jest znowu pierwszy kolor", Fore.RED)
wyswietl("To jest znowu drugi kolor", Fore.GREEN)
```

A poniżej rozwiązanie „kapsułkowe”, czyli obiektowe:

```
from lib_colorama import Fore, Back, Style

class Kolory(object):
  def __init__(self):
    self.kolor = Fore.RESET

  def wypisz(self, tekst):
    print self.kolor + tekst + Fore.RESET

obiekt1 = Kolory()
obiekt1.kolor = Fore.RED

obiekt2 = Kolory()
obiekt2.kolor = Fore.GREEN

obiekt1.wypisz('To jest pierwszy kolor')
obiekt2.wypisz('To jest drugi kolor')

obiekt1.wypisz('To jest znowu pierwszy kolor')
obiekt2.wypisz('To jest znowu drugi kolor')

# obiekt ma "w sobie" wszystkie potrzebne ustawienia formatowania tekstu
```

** Zadanie 1. Klasa Osoba

Stwórz klasę Osoba z danymi: imię, nazwisko, płeć, data urodzenia, metodą `przedstaw_sie`, wypisującą: „Nazywam się … i mam … lat", metodą `pomachaj` wypisującą: „… macha do Ciebie" oraz metodą `mrugnij` wypisującą: „… mruga do Ciebie po raz [pierwszy | drugi | trzeci | 4. | 5. | 6. itd.]".
Stwórz 3 osoby (czyli obiekty tej klasy), a następnie napisz program, w którym po kolei:

Pierwsza z nich przedstawi się, a następnie pomacha.
Druga z nich najpierw dwa razy pomacha, a później się przedstawi.
Trzecia mrugnie dwa razy, przedstawi się, po czym pomacha i mrugnie jeszcze raz.
Druga osoba mrugnie raz.
Wszystkie trzy osoby pomachają.

** Zadanie 2. Klasa Wartość

Stwórz klasę Wartość, która przechowuje wartość liczbową, którą można wypisać, do której można coś dodać i od której można coś odjąć. Napisz odpowiednie metody.

** Zadanie 3. Klasa ListaLiczb

Stwórz klasę ListaLiczb, która przechowuje listę liczb. Klasa udostępnia metody zwracające: średnią arytmetyczną, sumę liczb, liczbę elementów, element maksymalny, element minimalny, różnicę między elementem maksymalnym, a minimalnym. Do listy można dodać nowy element, można też usunąć element lub sprawdzić czy element istnieje na liście.
Bonus: Jeśli usuwamy element nieistniejący, program wypisuje błąd na czerwono, korzystając z obiektu klasy Kolory.

*** Zadanie 4. Grafika żółwiowa

Zapoznaj się z biblioteką turtle do grafiki żółwiowej (http://docs.python.org/library/turtle.html). Przy jej pomocy narysuj żółty trójkąt (niekoniecznie równoboczny i niekoniecznie wypełniony w środku) i czerwony okrąg. Jeśli do kółka użyłeś/użyłaś gotowej metody, zastanów się, jak je narysować bez niej.

*** Zadanie 5. Wahadło

Stwórz klasę Wahadło, dzięki której będzie można obliczać pozycję wahadła w kolejnych chwilach czasu.
Niech klasa zawiera:
konstruktor, w którym wczytuje się długość nitki oraz
dwie metody:
`odczytaj_pozycje` i
`przesun_czas`
Opis metod:
`odczytaj_pozycje()` zwraca dwuelementową listę `[x, y]` zawierającą współrzędne wahadła w danej chwili czasu (wg „czasu wahadła", który przestawia się jedynie przy pomocy metody `przesun_czas`)
`przesun_czas(ile_msec_do_przodu)` powoduje przesunięcie „czasu wahadła" o `ile_msec_do_przodu` (wartości ujemne są niepoprawne!)

*** Zadanie 6. Symulacja wahadła (kompilacja dwóch poprzednich zadań)

Korzystając z programów z dwóch poprzednich zadań stwórz symulację graficzną wahadła. Możesz też skorzystać z dowolnej innej znanej Ci biblioteki graficznej.
W razie pytań i wątpliwości zgłoś się do prowadzącego zajęcia.

Przydatne linki

- Dokumentacja Pythona: http://docs.python.org/ (dobra dokumentacja, kiepska wyszukiwarka; polecam wpisywać raczej w Google z dopiskiem: docs.python.org)
- Zadania do zaprogramowania, wraz z automatyczną sprawdzarką rozwiązań: http://pl.spoj.pl/problems/latwe/sort=-7 (otwiera sekcję: Łatwe, inne sekcje dostępne w menu)
- Najlepszy przyjaciel każdego programisty w razie problemów: Wujek Google ☺