Execuţia condiţionată

Expresii booleene

Expresia booleană este o expresie care este fie adevărată fie falsă. Operatorul == compară operanzii şi returnează True dacă aceştia sunt egali şi False altfel.

```
>>> 5 == 5
True
>>> 5 == 6
False
```

True şi False sunt valori speciale care aparţin clasei bool, nefiind de tip string.

```
>>> type(True)
<class 'bool'>
```

Ceilalţi operatori utilizaţi în comparaţii sunt:

!=, >, <, >=, <=, is, is not

Diferenţa între operatorii == şi is este aceea ca == se referă la valoare, iar is verifică dacă operanzii reprezintă acelasi obiect.

```
>>> a=[1,2,3]
>>> b=[1,2,3]
>>> a==b
True
>>> a is b
False
```

Operatori logici

Python are trei operatori logici: and, or, not.

Orice numar diferit de zero este interpretat ca fiind True.

```
>>> 17 and True
True
```

Instructiunea IF

Exemplu al instrucţiunii IF folosind două alternative:

If x%2 ==0:

print (‘ x este par’)

else:

print (‘x este impar’)

Condiţionare înlănţuită:

```
if x < y:
        print(‘x mai mic decat y’)
elif x > y:
        print('x mai mare decat y')
else:
        print('x si y sunt egale)
```

Elif este o abreviere de la “else if”, numarul instrucţiunilor elif nefiind limitat. Clauza else va fi întotdeauna plasată la final.

Condiţionare încuibarită:

```
if x == y:
        print('x and y are equal')
else:
        if x < y:
                print('x is less than y')
        else:
                print('x is greater than y')
```

Utilizarea excepţiilor prin clauzele try şi except

La executarea unor instrucţiuni pot apărea unele erori datorate introducerii eronate ale datelor de către utilizator, ceea ce determină o întrerupere automată a execuţiei.

Exemplu:

```
inp = input('Introduceti temperatura in grade Fahrenheit: ')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)
```

Dacă introducem o valoare invalidă, obţinem eroarea:

```
Introduceti temperatura in grade Fahrenheit:zece
Traceback (most recent call last):
File "temperatura.py", line 2, in <module>
```

fahr = float(inp)
ValueError: could not convert string to float: 'zece'

Python oferă support pentru tratarea excepţiilor prin instrucţiunea Try/except. Ideea este aceea de a încerca execuţia, iar în caz de eroare să fie acordată o alternativă

Exemplu:

```
inp = input('Introduceti temperatura in grade Fahrenheit: ')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Introduceti o valoare numerica!')
```

Exerciţii

1. Rescrieţi programul de calcul al salariului, acordând angajatului o rată orară de 1.5 ori mai mare pentru mai multe de 40 ore lucrate săptămânal.

Enter Hours: 45
Enter Rate: 10
Pay: 475.0

2. Rescrieţi programul de calcul al salariului folosind clauzele try şi except, astfel încât să fie gestionată introducerea unor caractere nenumerice printr-un mesaj de eroare.

Funcţii

Funţii predefinite

Python oferă o serie de funcţii deja construite, ce pot fi apelate fără a fi nevoie să fie definite. Acestea se referă la anumite probleme comune de calculat.

Exemple:

```
>>> max('Hello world')
'w'
>>> min('Hello world')
' '

>>> len('Hello world')
11
```

Lista funcţiilor predefinite din Python poate fi accesată la linkul următor:

https://docs.python.org/3/library/functions.html

Funcţii matematice

Python are un modul matematic ce oferă cele mai familiar funcţii. Pentru a putea utiliza acest modul, el trebuie importat sub forma:

```
>>> import math
```

Pentru a apela funcţiile incluse în modul, vom utiliza formatul *math.nume_functie().*

Exemplu:

```
print(math.sqrt(2) / 2.0)
```

Numere aleatoare

Funcţia random returneaza un număr de tip float între 0.0 şi 1.0.

Există mai multe funcţii random asociate modulului random.

Exemple:

```
import random

x = random.random()
print(x)

y = random.randint(5, 10)

print(y)


t = [1, 2, 3]

print(random.choice(t))
```

Adăugarea de noi funcţii

Atunci când dorim să creăm funcţionalităţi care nu sunt predefinite în Python, avem posibilitatea de a crea noi funcţii. Definiţia unei funcţii se face specificând numele funcţiei şi secvenţa de instrucţiuni ce urmează a se executa atunci când funcţia este apelată. Odată definită, funcţia poate fi reutilizată pe parcursul programului.

Exemplu:

```
def print_lyrics():
        print("I'm a lumberjack, and I'm okay.")
        print('I sleep all night and I work all day.')
```

Sintaxa pentru apelarea funcţiei este aceeaşi cu cea a funcţiilor predefinite.

```
>>> print_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Odata definite o funcţie, aceasta poate fi utilizată în interiorul altei funcţii.

Exemplu:

```
def repeat_lyrics():
        print_lyrics()
        print_lyrics()
```

Pentru a fi siguri ca o funcţe este definită înainte de a fi utilizată pentru prima data, trebuie avut în vedere fluxul execuţiei programului. Trebuie să reţinem că instrucţiunile din interiorul unei funcţii nu sunt executate până la prima apelare a funcţiei.

Parametri şi argumente

Unele din funcţiile predefinite necesită argumente pentru a fi executate. De exemplu apelarea funcţiei math.sin necesită un număr ca şi argument. În interiorul funcţiei argumentle sunt asociate unor variabile, numite parametri.

Exemplu:

```
def print_twice(nume):
        print(nume)
        print(nume)
```

Funcţia asociază argumentul către un parametru denumit nume. Când funcţia este apelată, afişează valoarea parametrului, de două ori.

```
>>>print_twice('Ana')
Ana
Ana
>>> print_twice(17)
```

```
17
17
```

Funcţii productive şi funcţii vide

Unele dintre funcţiile pe care le utilizăm returnează rezultate (denumite funcţii productive), cum sunt funcţiile matematice din exemplele de mai sus. Altele, cum este funcţia print_twice execută instrucţiunile dar nu returnează o valoare. Acestea se numesc funcţii vide.

Dacă într-un program nu asociem o funcţie productivă unei variabile, valoarea acesteia se pierde.

Exemplu:

Apelarea

math.sqrt(5)

nu produce niciun fel de rezultate. Pentru a deveni utilă, vom folosi:

x=math.sqrt(5)

în acest fel valoarea funcţiei se va asocia variabilei.

Dacă asociem unei variabile o funcţie vidă, nu vom obţine nimic.

Exemplu:

```
>>> result = print_twice('Bing')
Bing
Bing
>>> print(result)
None
```

Pentru ca o funcţie să returneze un rezultat, vom folosi instrucţiunea return în cadrul acesteia.

Exemplu:

```
def adunare(a, b):
        suma = a + b
        return suma
x = adunare(3, 5)
```

```
print(x)
```

Exerciţii

Rescrieţi programul de calcul al salariului in varianta cu plata de 1.5 rata normal pentru timp de lucru de peste 40 ore, folosind functia calcul_salariu(ore, tarif_orar).

Iteraţiile

Actualizarea variabilelor

În Python actualizarea prin incrementare a variabilelor se face după urmatoarea sintaxă:

```
x=0
x=x+1
```

sau

```
x+=1
```

Instrucţiunea While

Executarea repetitivă a unor instrucţiuni este un element comun în limbajele de programare.

Pentru instrucţiunea While sintaxa poate fi dedusă din exemplul următor:

```
n = 5
while n > 0:
        print(n)
        n = n - 1
print('Terminat!')
```

În cazul în care condiţia de terminare a ciclului de execuţie nu ar fi îndeplinită, instrucţiunea ar genera o buclă infinită.

Exemplul următor presupune o buclă infinită deoarece expresia din instrucţiunea while este constanta True:

```
n = 10
while True:
        print(n, end=' ')
        n = n - 1
print('Terminat!')
```

Dacă veţi face greşeala să rulaţi acest program, veţi observa că programul nu poate fi oprit decât forţat.

Nu toate ciclurile infinite sunt disfuncţionale ca în acest exemplu. În situaţia în care nu cunoaştem de la început de câte ori are nevoie utilizatorul să introducă valori, putem genera intenţionat o buclă infinită, care să se oprească la un cuvânt cheie introdus.

```
while True:
        line = input('> ')
        if line == 'gata':
                break
        print(line)
print('Terminat!')
```

Instrucţiunea break întrerupe bucla infinită la introducerea cuvântului gata de către utilizator.

Uneori, într-un ciclu repetitive este nevoie de a încheia bucla şi a continua cu următoarea iteraţie. În această situaţie putem utiliza instrucţiunea continue.

Exemplu:

```
while True:
        linie = input('> ')
        if linie[0] == '#':
                continue
        if linie == 'gata':
                break
        print(linie)
print('Gata!')
```

Programul de mai sus nu va afisa linia care incepe cu #, fara a intrerupe executia ciclului.

Iteraţii cu număr definit de paşi

Uneori este necesar să realizăm o iteraţie într-o mulţime de cuvinte, linii într-un fisier sau o listă de numere. Sintaxa instrucţiunii for poate fi dedusă din exemplul următor:

```
prieteni = ['Alina', 'Ionel', 'Sandu']
```

```
for prieten in prieteni:
        print('La multi ani:', prieten)
print('Gata!')
```

Variabila prieteni este o lista de trei siruri de caractere iar ciclul for parcurge lista şi executa pentru fiecare pas corpul instructiunii, rezultand:

La multi ani: Alina

La multi ani: Ionel

La multi ani: Sandu

Gata!

Numărarea elementelor într-o listă se face în modul următor:

```
count = 0
for i in [3, 41, 12, 9, 74, 15]:
        count = count + 1
print('Count: ', count)
```

Un mod echivalent de a scrie instrucţiunea for cu alte limbaje de programare este:

```
for i in range(10):
        print (i)
```

Funcţia range(10) returnează o listă cu elemente intregi de la 0 la 9.

Calcularea maximului si a minimului dintr-o lista

Testaţi funcţionarea programului de mai jos iar apoi modificati programul astfel incat sa calculeze minimul in mod similar.

```
largest = None
print('Before:', largest)
for itervar in [3, 41, 12, 9, 74, 15]:
        if largest is None or itervar > largest :
                largest = itervar
        print('Loop:', itervar, largest)
print('Largest:', largest)
```

Exerciţiu:

Scrieţi un program care citeste de la utilizator numere pana cand acesta introduce “gata”. Dupa introducerea acestui cuvant vor fi afisate suma numerelor, numarul de valori introduse si media lor. Daca utilizatorul introduce altceva decat un numar, acest lucru va fi detectat prin utilizarea try/except si se va afisa un mesaj de eroare, iar programul va solicita introducerea unui nou numar, fara a opri executia.

Bibliografie:

Charles R. Severance, Python for Everybody – Exploring Data Using Python 3, 2016, www.py4e.com