Propunător: Ștefănescu Daniela Narcisa

ISJ Suceava

Structuri repetitive in Python

Structura repetitive while. Exemple
Structura repetitive for. Exemple
Aplicații

Structura repetitivă `while`

Structura `while` ne permite executarea unui bloc de cod în mod repetat *cât timp* o condiţie este satisfăcută. O condiţie poate fi orice expresie care returnează o valoare `bool` (adevărat/fals) la fel ca în cazul structurii `if` de mai sus. O condiţie este considerată satisfăcută atunci când îşi schimbă valoarea în `False`. Blocul de instrucţiuni va fi imediat sub structura `while`, iar în Python este nevoie să fie indentat cu patru spaţii (în alte limbaje ar putea fi delimitat de acolade `{}`).

Exemple de probleme ce necesită bucle repetitive: câte numere pare sunt într-o listă de elemente, pornind de la o coadă de clienţi, serveşte fiecare client până când s-a golit coada.

De exemplu:

```python
x = 5
while x > 0:
    print(x)
    x -= 1  # prescurtare pentru x = x - 1
```

Python

Acest cod ne va afişa:

```
5
4
3
2
1
```

Python

Haideţi să vedem mai de aproape ce se întâmplă în această bucată de cod. Primul lucru pe care îl facem este să declarăm o nouă variabilă care va avea rolul de a păstra progresul nostru după fiecare pas al buclei repetitive. Valoarea `5` a fost aleasă aleator, puteţi să o schimbaţi cum vreţi.

Pentru a începe bucla repetitivă scriem cuvântul `while` urmat de condiţia de oprire. Blocul de cod subordonat acestei structuri va fi executat până când condiţia devine *FALSĂ*. Cuvântul `while` poate fi

tradus prin "*cât timp*", iar structura de mai sus se poate citi astfel: *cât timp variabila `x` este mai mare strict decât `0`, execută codul de mai jos*.

Blocul de instrucţiuni începe prin a afişa pe ecran valoarea variabilei `x` folosind funcţia `print()`, iar la final decrementează valoarea variabilei `x`. Această decrementare este extrem de importantă deoarece ea ne apropie de satisfacerea condiţiei. Fără o metodă de a ne apropia de finalul buclei (de schimbarea valorii condiţiei în `False`) nu vom putea ieşi din această buclă.

Starea în care ne vom afla în cazul în care condiţia de ieşire nu este satisfăcută niciodată se numeşte **ciclu infinit** sau **buclă infinită**. Atunci când ajungeţi într-un ciclu infinit, codul vostru nu va mai avansa şi va repeta execuţia aceluiaşi bloc de cod aflat în bucla repetitivă la nesfârşit (sau până la epuizarea resurselor unde e cazul). Pentru a ieşi dintr-un ciclu repetitiv în mod forţat puteţi să apăsaţi tastele `CTRL+C` sau un buton de oprire a execuţiei (dacă există în IDE-ul folosit de voi).

Mai mult, decrementarea nu este întotdeauna metoda potrivită pentru a ieşi din buclă. Dacă începem cu valoarea lui `x = 1` în loc de `5` şi modificăm condiţia precedentă în `x <= 5`, atunci decrementarea nu ar fi făcut decât să ne depărteze de satisfacerea condiţiei. În acest caz am fi avut nevoie de incrementarea valorii:

```python
x = 1
while x <= 5:
    print(x)
    x += 1  # prescurtare pentru x = x + 1
```
Python

Rezultatul execuţiei va fi invers faţă de cel de mai sus deoarece de această dată `x` începe de la `1` şi se termină când are o valoare mai mare decât `5`.

Atunci când vă confruntaţi cu o buclă repetitivă `while`, veţi avea două decizii de luat: când se opreşte bucla şi cum vă puteţi apropia de satisfacerea condiţiei. În majoritatea cazurilor incrementarea sau decrementarea variabilei (nu neapărat cu valoarea `1`) este suficientă, însă pot apărea şi soluţii mai complexe.

Structura repetitivă `for`

Pe lângă bucla repetitivă `while`, în Python, mai este disponibilă şi structura `for` pentru crearea de bucle repetitive.

Această structură ne permite să executăm câte un pas pentru fiecare element dintr-un iterabil dat (în toate exemplele noastre acel iterabil va fi privit drept o listă sau un dicţionar). Procesul de a parcurge elementele unei liste în vederea executării anumitor operaţii la fiecare pas mai poartă numele de *iterare peste listă*. De aici si termenul de *iterabil*. Un iterabil este orice fel de obiect peste care putem itera (parcurge elementele într-o buclă).

De exemplu:

```python
for x in range(1, 6):
    print(x)
```
Python

În urma executării acestui cod se va afişa pe ecran

```
1
2
3
4
5
```

Python

Această bucată de cod este echivalentă cu ultimul exemplu de mai sus. Funcţia `range()` (în traducere: *interval*) ne permite crearea unui interval închis la capătul din stânga şi deschis la capătul din dreapta. În exemplul nostru, se creează intervalul `[1, 6)`. Dacă este specificat un singur parametru, această funcţie va presupune că începe de la `0`. Adică `range(6)` creează intervalul `[0, 6)`.

Să ne uităm acum mai de aproape la exemplul de mai sus. Observaţi cum, de această dată, nu am mai avut nevoie să declarăm variabila `x` înainte de structura noastră. Această variabilă va fi creată cu o nouă valoare la fiecare pas din bucla `for`. La fiecare pas din buclă, variabila `x` va avea următoarea valoare din intervalul `[1, 6)` şi va afişa pe ecran acea valoare.

Bucla `for` se va încheia atunci când a fost parcurs întreg iterabilul (`x` a avut fiecare valoare din intervalul `[1, 6)`). Din acest motiv nu este nevoie să avem o metodă de a ne asigura că bucla noastră îşi satisface condiţia. Acea condiţie este garantată să se îndeplinească odată cu epuizarea elementelor iterabilului (listei, intervalului de numere).

Bucla `for` nu poate intra într-o buclă infinită decât dacă lista este infinită (este posibilă crearea unei astfel de structuri, dar nu ne interesează acest lucru pe moment).

Exemplul de mai sus se poate scrie şi cu o listă simplă în locul funcţiei `range()`:

```
for x in [1, 2, 3, 4, 5]:
    print(x)
```

Python

şi va avea exact acelaşi rezultat ca mai sus:

```
1
2
3
4
5
```

Python

Concluzii:

Toate structurile prezentate în această lecţie se încheie prin două puncte "`:`" şi sunt urmate de un bloc de instrucţiuni indentat cu patru spaţii.

În alte limbaje de programare precum Java, C#, C++, există şi structura `do-while` care este inversul structurii `while`. Această structură poate fi citită astfel: *repetă blocul de cod până când condiţia devine adevărată*. În cazul buclei `while`, repetăm execuţia *CÂT TIMP* condiţia este adevărată (deci ne oprim când ea devine falsă).

Orice buclă `while` poate fi rescrisă cu o buclă `for` şi invers, însă nu este întotdeauna uşor de făcut. Ca regulă generală, atunci când vrem să parcurgem o listă vom folosi bucle `for`, iar atunci când vrem să executăm un bloc în funcţie de o anumită condiţie, folosim `while`.

Atât pentru astăzi, vom învăţa mult mai multe despre aceste structuri în timp ce le folosim în exemplele şi exerciţiile ce vor urma. Dacă aveţi orice fel de întrebări, nu ezitaţi să scrieţi în comentarii. Dacă nu, pe data viitoare.

Aplicații propuse

1. **Sa se scrie si sa se apeleze intr-un program demonstrativ o functie pentru afisarea echivalentului binar al unui numar natural dat. Exemple: 11=1011, 25 =11001, 23= 10111.**

Se imparte numarul la 2 , pina ajunge la 0. Se retin resturile, dar in ordine inversa. Ele se pot adauga intr-

o lista si lista afisata invers. (r = n% 2, lista.append(r)). Exista functia reverse pentru asta

```
lista=[]
while n!=0:
    r=n%2
   n=n//2
  lista.append(r)
```

Eu va implementez recursiv

11 = 2***5** +1

5 = 2***2** +1

2 = 2***1**+0

1 = 2*0 +1

```
def transformare_baza2(n):
   if n!=0:
      r = n%2
      transformare_baza2(n//2)
```

```
        print(r,end="")


    n = int(input("Dati n:"))
    print("n2=", end =" ")
    transformare_baza2(n)
```

2. Să se calculeze expresia S=1-2+3-4+....+n

```
n = int(input("Dati n:"))
S = 0
for i in range(1,n+1):
   if i%2==0:
      S = S -i
   else:
      S = S+i
print("S =",S)
```

3. Se citeşte un număr natural n. Să se determine cifra maximă din număr, folosind un subprogram. Exemplu: Dacă n=7934 , atunci cifra maximă este 9.

```
def cifra_maxima(x):
   Max = 0
   while x>0:
      c = x%10
      x = x//10
      if c>Max:
         Max =c
   return Max


n = int(input("Numarul:"))
```

```
print("Cifra maxima din ", n, "este", cifra_maxima(n))
```

4 **Se introduce de la tastatură un număr întreg n. Se cere să se calculeze şi să se afişeze:**

a) media aritmetică a cifrelor pare;

b) produsul cifrelor impare

```
n = int(input("n="))
s = 0
k = 0
p = 1

# la fiecare pas extrag ultima cifra, apoi o sterg din numarul n
# folosesc cifra curenta

while n>0:
    c = n%10
    n = n//10
    if c%2 ==0:
        s = s + c
        k = k +1
    else:
        p = p * c
print("Media cifrelor pare:", s//k)
print("Produsul cifrelor impare", p)
```

5. Se citesc m, n două variabile întregi pozitive.

- Să se determine toate pătratele perfecte cuprinse între m şi n, inclusiv.

- Să se determine toate numerele prime cuprinse între m şi n.

- Să se determine toate numerele de 4 cifre care se divid atît cu n cît şi cu m.

- Să se determine c.m.m.d.c. al celor două numere folosind algoritmul lui Euclid.

Scrieti un program care sa rezolve cerintele enuntate mai sus prin implementarea unui meniu cu urmatoarele optiuni:

1. citire + afisare numere (2 functii)

2. patrate perfecte

3. nr. prime

4. nr. divizibile cu n si m

5. cmmdc

6. exit

'''

```
import math
def alege_optiune():
    print('1 - Citire + afisare numere')
    print('2 - Patrate perfecte')
    print('3 - Nr. prime')
    print('4 - Nr. divizibile cu n si m')
    print('5 - Cmmdc')
    print('6 - Termina program')


def citire():
    m = int(input('m:'))
    n = int(input('n(n >m):'))
    return m,n


def afisare_perfecte():


    i = m
```

```
    while i <= n:
        j = int(math.sqrt(i))
        if j*j ==i:
            print(j*j,end =" ")
        i = i+1


def prim(num):
    if num ==1:
        ok = 0
    elif num > 1:
        ok = 1
        for i in range(2,num//2):
            if (num % i) == 0:
                ok =0
                break
    return ok


def cmmd(a,b):
    while a!=b:
        if a>b:
            a -= b
        else:
            b -=a
    return a


n = 0
m = 0
while 1:
```

```
print()
alege_optiune()
opt = int(input('Alegeti o optiune:'))
if opt ==1:
    m ,n = citire()
if opt == 2:
    afisare_perfecte()
if opt == 3:
    print("Numerele prime:"," ")
    for i in range(m,n+1):
        if prim(i):
            print(i, end=" ")
if opt == 4:
    print("Numerele divizibile",end =" ")
    for i in range(1000, 10000):
        if i%n==0 and i%m==0:
            print(i,end=" ")
if opt == 5:
    print("Cmmdc:", cmmd(n,m))
if opt==6:
    break
```