LAB 2.Structuri de control condiționale. Structuri repetitive. Module - math, random, datetime, time.

Obiective:

Studenții trebuie:

- Să înțeleagă și să aplice **structurile condiționale** (if, elif, else) pentru luarea deciziilor în programe.
- Să utilizeze **structurile repetitive** (while și for) pentru a repeta acțiuni.
- Să importe și să folosească **module utile** precum math, random, datetime și time.
- Să rezolve probleme practice care implică **luarea deciziilor și repetiția codului**.

I. Condiții Python și instrucțiuni If

Python acceptă condițiile logice obișnuite din matematică:

- Este egal cu: a == b
- Nu este egal cu: a != b
- Mai puțin decât: a < b
- Mai mic sau egal cu: a <= b
- Mai mare decât: a > b
- Mai mare sau egal cu: a >= b

Aceste condiții pot fi utilizate în mai multe moduri, cel mai frecvent în „instrucțiuni if” și bucle.

O „instrucțiune if” este scrisă folosind cuvântul cheie if .

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

În acest exemplu folosim două variabile, a și b , care sunt utilizate ca parte a instrucțiunii if pentru a testa dacă b este mai mare decât a . Deoarece a este 33 și b este 200 , știm că 200 este mai mare decât 33 și astfel imprimăm pe ecran că „b este mai mare decât a”.

Indentare

Python se bazează pe indentare (spații albe la începutul unei linii) pentru a defini domeniul de aplicare în cod. Alte limbaje de programare folosesc adesea paranteze în acest scop.

Exemplu

Instrucțiunea If, fără indentare (va genera o eroare):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

Elif

Cuvântul cheie elif este modul Python de a spune „dacă condițiile anterioare nu erau adevărate, atunci încercați această condiție”.

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

În acest exemplu , a este egal cu b , deci prima condiție nu este adevărată, dar condiția elif este adevărată, așa că imprimăm pe ecran că „a și b sunt egale”.

Altfel

Cuvântul cheie else prinde orice nu este prins de condițiile precedente.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

În acest exemplu , a este mai mare decât b , deci prima condiție nu este adevărată, de asemenea, condiția elif nu este adevărată, așa că mergem la condiția else și imprimăm pe ecran că „a este mai mare decât b”.

Puteți avea și else fără elif:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Mâna scurtă IF

Dacă aveți o singură instrucțiune de executat, o puteți pune pe aceeași linie cu instrucțiunea if.

```
if a > b: print("a is greater than b")
```

Mâna Scurtă Dacă... Else

Dacă aveți o singură instrucțiune de executat, una pentru if și una pentru else, puteți pune totul pe aceeași linie:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

Această tehnică este cunoscută sub numele de Operatori Ternari sau Expresii Condiționale .

De asemenea, puteți avea mai multe instrucțiuni else pe aceeași linie:

Exemplu

O linie if else declarație, cu 3 condiții:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

Şi

Cuvântul cheie and este un operator logic și este folosit pentru a combina instrucțiuni condiționale:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Sau

Cuvântul or cheie este un operator logic și este folosit pentru a combina instrucțiuni condiționale:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Nu

Cuvântul notcheie este un operator logic și este folosit pentru a inversa rezultatul instrucțiunii condiționate:

```
a = 33
b = 200
if not a > b:
  print("a is NOT greater than b")
```

IF Imbricat

Puteți avea ifinstrucțiuni în interiorul ifinstrucțiunilor, aceasta se numește instrucțiuni *imbricate* if .

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Declarația de trecere

ifdeclarațiile nu pot fi goale, dar dacă dintr-un motiv oarecare aveți o ifdeclarație fără conținut, introduceți passdeclarația pentru a evita să obțineți o eroare.

```
a = 33
b = 200

if b > a:
  pass
```

II. Bucle Python

Python are două comenzi de buclă primitive:

- bucle while
- For bucle

Bucla while

Cu bucla while putem executa un set de instrucțiuni atâta timp cât o condiție este adevărată.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Bucla while necesită ca variabilele relevante să fie pregătite, în acest exemplu trebuie să definim o variabilă de indexare, i , pe care o setăm la 1.

Declarația break

Cu instrucțiunea break putem opri bucla chiar dacă condiția while este adevărată:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Declarația de continuare

Cu instrucțiunea continue putem opri iterația curentă și continua cu următoarea:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Declarația else

Cu instrucțiunea else putem rula un bloc de cod o dată când condiția nu mai este adevărată:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Python FOR

O buclă for este folosită pentru iterarea unei secvențe (adică fie o listă, un tuplu, un dicționar, un set sau un șir).

Acesta seamănă mai puțin cu cuvântul cheie for din alte limbaje de programare și funcționează mai mult ca o metodă iteratoare așa cum se găsește în alte limbaje de programare orientate pe obiecte.

Cu bucla for putem executa un set de instrucțiuni, o dată pentru fiecare element dintr-o listă, tuplu, set etc.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Bucla for nu necesită o variabilă de indexare care să fie setată în prealabil.

Buclă printr-un șir

Chiar și șirurile sunt obiecte iterabile, ele conțin o secvență de caractere:

```
for x in "banana":
  print(x)
```

Declarația break

Cu instrucțiunea break putem opri bucla înainte ca aceasta să fi parcurs toate elementele:

ex. Ieșiți din buclă când xeste „banana”:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

ex. Ieșiți din buclă când xeste „banana”, dar de data aceasta pauză vine înainte de imprimare:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Declarația de continuare

Cu instrucțiunea continue putem opri iterația curentă a buclei și putem continua cu următoarea:

Exemplu

Nu tipăriți banane:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Funcția range().

Pentru a trece printr-un set de cod de un anumit număr de ori, putem folosi funcția range() ,

Funcția range () returnează o secvență de numere, începând de la 0 în mod implicit și se incrementează cu 1 (în mod implicit) și se termină la un număr specificat.

Exemplu

Folosind funcția range():

```
for x in range(6):
  print(x)
```

Rețineți că intervalul (6) nu este valorile de la 0 la 6, ci valorile de la 0 la 5.

Funcția range () este implicit 0 ca valoare de pornire, totuși este posibil să specificați valoarea de pornire adăugând un parametru: range(2, 6) , ceea ce înseamnă valori de la 2 la 6 (dar fără să includă 6):

```
for x in range(2, 6):
  print(x)
```

Funcția range () în mod implicit crește secvența cu 1, totuși este posibil să specificați valoarea de increment adăugând un al treilea parametru: range(2, 30, 3) :

```
for x in range(2, 30, 3):
  print(x)
```

Altfel în bucla For

Cuvântul elsecheie într-o forbuclă specifică un bloc de cod care trebuie executat când bucla este terminată:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Notă: Blocul elseNU va fi executat dacă bucla este oprită de o breakinstrucțiune.

Rupe bucla când xeste 3 și vezi ce se întâmplă cu elseblocul:

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

Bucle imbricate

O buclă imbricată este o buclă în interiorul unei bucle.

„Bucla interioară” va fi executată o dată pentru fiecare iterație a „buclei exterioare”:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Declarația de trecere

forbuclele nu pot fi goale, dar dacă dintr-un motiv oarecare aveți o forbuclă fără conținut, introduceți passinstrucțiunea pentru a evita o eroare.

```
for x in [0, 1, 2]:
  pass
```

📌 Probleme de Programare – If, For, Else, While și Tipuri de Variabile

Mai jos sunt 5 probleme pentru fiecare nivel de complexitate. Alegeti o problema din fiecare nivel

---

🔹 Nivel 1: Începător (Probleme simple)

1️⃣ **Număr pozitiv, negativ sau zero**

- Cere utilizatorului un număr și verifică dacă este **pozitiv, negativ sau zero**.

2️⃣ **Afișare numere pare până la N**

- Utilizatorul introduce un număr N, iar programul afișează toate numerele pare de la 1 la N folosind for.

3️⃣ **Suma primelor N numere naturale**

- Utilizează for pentru a calcula suma primelor N numere naturale introduse de utilizator.

4️⃣ **Numărare litere mari și mici**

- Cere utilizatorului un text și numără câte litere **mari** și câte litere **mici** conține.

5️⃣ **Afișarea unui mesaj de N ori**

- Citește un număr N și afișează mesajul "Python este cool!" de N ori folosind while.

🔹 Nivel 2: Mediu (Probleme mai complexe)

6️⃣ **Ghicirea unui număr secret**

- Generează un număr între 1 și 20, iar utilizatorul trebuie să-l ghicească.
- Programul oferă indicii ("mai mare" sau "mai mic") și se repetă până când ghicește corect.

7️⃣ **Numere prime între 1 și N**

- Cere utilizatorului un număr N și afișează toate **numerele prime** până la N.

8️⃣ Căutare într-o listă

- Creează o listă de 5 nume și cere utilizatorului să introducă un nume.
- Verifică dacă numele se află în listă sau nu și afișează un mesaj corespunzător.

9️⃣ Cel mai mare și cel mai mic număr dintr-o listă

- Citește N numere introduse de utilizator și determină **valoarea minimă și maximă**.

🔟 Numere Fibonacci

- Afișează primele N numere din **șirul Fibonacci** folosind for sau while.

🔹 Nivel 3: Avansat (Probleme mai dificile)

1️⃣1️⃣ Verificare palindrom

- Cere un cuvânt și verifică dacă este **palindrom** (se citește la fel înainte și înapoi).

1️⃣2️⃣ Divizori ai unui număr

- Citește un număr N și afișează toți **divizorii** săi.

1️⃣3️⃣ Factorialul unui număr

- Cere un număr N și calculează **factorialul** său folosind for sau while.

1️⃣4️⃣ Numărare vocale și consoane

- Citește un text și numără câte **vocale** și **consoane** conține.

1️⃣5️⃣ Joc: "Par sau Impar"

- Programul generează un număr aleatoriu între 1 și 100.
- Utilizatorul trebuie să ghicească dacă este **par sau impar**.