**Ministerul Educației și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**RAPORT**

Lucrare de laborator nr.4

la cursul „Programarea Declarativă”

**Tema 4:** „ Introducere în Python. Definirea și apelarea funcțiilor de ordin înalt map(), filter(), reduce() și a funcțiilor anonime - lambda(”

A efectuat : **st. gr. TI-214 Buza Cătălin**

A verificat:  **M.Rusu**

**Chișinău 2023**

**Exercițiul 1:**

Definiți un dicționar ce folosește șiruri de caractere pe post de chei și elemente float pe post

de valori. Afișați doar cheile dicționarului și mai apoi tupluri formate din chei și valori.

print("Exercitiul 1\n")

# Crearea dicționarului

dictionar = {

"cheie1": 3.14,

"cheie2": 2.71,

"cheie3": 1.618,

"cheie4": 0.577

}

# Afișarea cheilor dicționarului

chei = dictionar.keys()

print("Chei:", chei)

# Afișarea tuplurilor (cheie, valoare)

tupluri = [(cheie, dictionar[cheie]) for cheie in dictionar]

print("Tupluri (cheie, valoare):", tupluri)

- Se începe prin a crea un dicționar numit `dictionar` care conține chei (string-uri) asociate cu valori (numere float).

- Se afișează lista de chei folosind `dictionar.keys()`.

- Se construiește o listă de tupluri (cheie, valoare) folosind o list comprehension, care iterează prin cheile dicționarului și extrage valorile corespunzătoare.

**Exercițiul 2**

Generați un număr aleator între 0 și 10000, ce reprezintă un număr de secunde. Calculați reprezentarea numărului de secunde în ore, minute și secunde și afișați rezultatul formatat sub forma hh:mm:ss. Alternativ, folosiți modulul datetime

print("Exercitiul 2\n")

import random

# Generați un număr aleator între 0 și 10000 (numărul de secunde)

numar\_de\_secunde = random.randint(0, 10000)

# Calculați orele, minutele și secundele

ore = numar\_de\_secunde // 3600

minute = (numar\_de\_secunde % 3600) // 60

secunde = numar\_de\_secunde % 60

# Formatați rezultatul sub forma "hh:mm:ss"

timp\_formatat = "{:02d}:{:02d}:{:02d}".format(ore, minute, secunde)

print(f"Numărul aleator de secunde: {numar\_de\_secunde}")

print(f"Reprezentare formatată: {timp\_formatat}")

- Aici se utilizează modulul `random` pentru a genera un număr aleator de secunde între 0 și 10000.

- Se calculează orele, minutele și secundele folosind diviziuni și modulo.

- Rezultatul este formatat sub forma "hh:mm:ss" utilizând un șir de formatare.

**Exercițiul 3**

Definiți două obiecte de tip float și calculați suma, diferența lor utilizand funcții lamda(). Apelați toate funcțiile din listă pe rând și cu alte tipuri de date, folosiți funcția map().

print("Exercitiul 3\n")

# Definirea celor două obiecte de tip float

float1 = 5.5

float2 = 3.3

# Definirea funcțiilor lambda pentru suma și diferența

suma = lambda x, y: x + y

diferenta = lambda x, y: x - y

# Crearea unei liste cu cele două funcții lambda

functii\_lambda = [suma, diferenta]

# Definirea altor tipuri de date

int1 = 2

int2 = 1

# Utilizarea funcției map pentru a apela funcțiile lambda pe diferite tipuri de date

rezultate = []

for functie in functii\_lambda:

rezultate\_float = list(map(lambda f: f(float1, float2), [functie]))

rezultate\_int = list(map(lambda f: f(int1, int2), [functie]))

rezultate.append((rezultate\_float, rezultate\_int))

# Afișarea rezultatelor

for i, functie in enumerate(functii\_lambda):

print(f"Rezultate pentru funcția {functie.\_\_name\_\_}:")

print(f"Float: {rezultate[i][0][0]:.2f}")

print(f"Int: {rezultate[i][1][0]}")

- În acest exercițiu, se definesc două obiecte de tip float (`float1` și `float2`).

- Se definesc două funcții lambda (`suma` și `diferenta`) pentru a efectua operații de adunare și scădere.

- Se creează o listă numită `functii\_lambda` care conține cele două funcții definite anterior.

- Se definesc și alte două variabile de tip întreg (`int1` și `int2`).

- Se utilizează funcția `map()` pentru a apela funcțiile lambda pe diferite combinații de tipuri de date.

**Exercițiul 4**

Definiți o listă de valori întregi și afișați doar valorile distincte din aceasta (folosiți funcția filter()).

print("Exercitiul 4\n")

# Definirea listei de valori întregi

valori\_intregi = [1, 2, 2, 3, 4, 4, 5, 6, 6, 7]

# Utilizarea funcției filter pentru a obține valorile distincte

valori\_distincte = list(filter(lambda x: valori\_intregi.count(x) == 1, valori\_intregi))

# Afișarea valorilor distincte

print("Valorile distincte din lista sunt:", valori\_distincte)

- Aici se definește o listă de valori întregi (`valori\_intregi`).

- Se folosește funcția `filter()` pentru a obține valorile distincte. Se folosește o funcție lambda care verifică de câte ori apare fiecare valoare în listă.

**Exercițiul 5**

Definiți o funcție ce returnează numărul de apariții ale unui caracter într-un string (folosiți funcția reduce()).

print("Exercitiul 5\n")

from functools import reduce

def numar\_aparitii\_caracter(sir, caracter):

# Utilizăm funcția reduce pentru a număra aparițiile caracterului

numar\_aparitii = reduce(lambda acc, char: acc + 1 if char == caracter else acc, sir, 0)

return numar\_aparitii

# Exemplu de utilizare

text = "Lorem ipsum dolor sit amet"

caracter\_cautat = 'z'

rezultat = numar\_aparitii\_caracter(text, caracter\_cautat)

print(f"Caracterul '{caracter\_cautat}' apare de {rezultat} ori în text.")

- Aici se definește o funcție numită `numar\_aparitii\_caracter` care primește un șir de caractere (`sir`) și un caracter (`caracter`).

- Funcția folosește funcția `reduce()` pentru a număra aparițiile caracterului în șir. Este utilizată o funcție lambda pentru a actualiza contorul.

- Se afișează rezultatul.

**Exercițiul 6**

Definiți o funcție ce concatenează oricâte stringuri sunt date la intrarea sa.

print("Exercitiul 6\n")

def concateneaza\_stringuri(\*args):

# Utilizăm funcția join() pentru a concatena toate șirurile date ca argumente

rezultat = "".join(args)

return rezultat

# Exemplu de utilizare

str1 = "Hello, "

str2 = "world!"

str3 = " This is a concatenated string."

rezultat = concateneaza\_stringuri(str1, str2, str3)

print(rezultat)

- Aici se definește o funcție numită `concateneaza\_stringuri` care primește un număr variabil de argumente de tip șir de caractere (`\*args`).

- Funcția folosește funcția `join()` pentru a concatena toate șirurile primite ca argumente.

- Se afișează rezultatul.

**Exercițiul 7**

Definiți o listă de funcții lambda ce returnează: al doilea caracter dintr-un string; stringul cu litere majuscule; poziția pe care se găsește un anumit caracter dat la intrare. Apelați toate funcțiile din listă pe rând.

print("Exercitiul 7\n")

# Definirea listei de funcții lambda

functii\_lambda = [

lambda x: x[1] if len(x) >= 2 else None, # Returnează al doilea caracter

lambda x: x.upper(), # Convertște șirul în litere majuscule

lambda x, c: x.find(c) # Găsește poziția caracterului dat

]

# Exemplu de șir de test

text = "exemplu"

# Apelarea funcțiilor din listă

for functie in functii\_lambda:

if len(functie.\_\_code\_\_.co\_varnames) == 1: # Verificăm numărul de parametri ai funcției

print(f"Rezultat pentru funcția lambda: {functie(text)}")

else:

print(f"Rezultat pentru funcția {functie.\_\_name\_\_}: {functie(text, 'x')}")

- Aici se definește o listă de funcții lambda (`functii\_lambda`) care conține trei funcții lambda diferite.

- Fiecare funcție lambda primește un șir de caractere (`x`). Prima funcție returnează al doilea caracter, a doua funcție transformă șirul în litere majuscule, iar a treia funcție găsește poziția unui caracter dat (`'x'` în acest caz).

- Se testează apelarea acestor funcții cu un șir de test (`text`).

**Exercițiul 8**

Definiți o funcție ce calculează media a trei note sprecificate la intrare. Dacă la apel nu se trimit toate notele, se vor folosi valori implicite egale cu 4. Apelați funcția cu diferite combinații de argumente poziționale și keyword

print("Exercitiul 8\n")

def media\_notelor(nota1=4, nota2=4, nota3=4):

media = (nota1 + nota2 + nota3) / 3

return media

# Exemple de apeluri ale funcției cu diferite combinații de argumente

media1 = media\_notelor(8, 9, 7)

media2 = media\_notelor(6, 5) # A treia notă va fi implicit 4

media3 = media\_notelor() # Toate notele vor fi implicite 4

media4 = media\_notelor(nota3=9, nota1=7) # Specificăm notele folosind argumente de tip keyword

print("Media 1:", media1)

print("Media 2:", media2)

print("Media 3:", media3)

print("Media 4:", media4)

- Aici se definește o funcție numită `media\_notelor` care primește trei argumente cu valori implicite (`nota1=4`, `nota2=4`, `nota3=4`).

- Funcția calculează media acestor note și o returnează.

- Sunt prezentate exemple de apeluri ale funcției cu diferite combinații de argumente, inclusiv utilizarea argumentelor implicite.

**Exercițiul 9**

Definiți o funcție recursivă ce afișează suma primelor N numere naturale.

print("Exercitiul 9\n")

def suma\_primele\_N\_naturale(N):

if N <= 0:

return 0

else:

return N + suma\_primele\_N\_naturale(N - 1)

# Exemplu de apel al funcției pentru suma primelor 5 numere naturale

N = 100

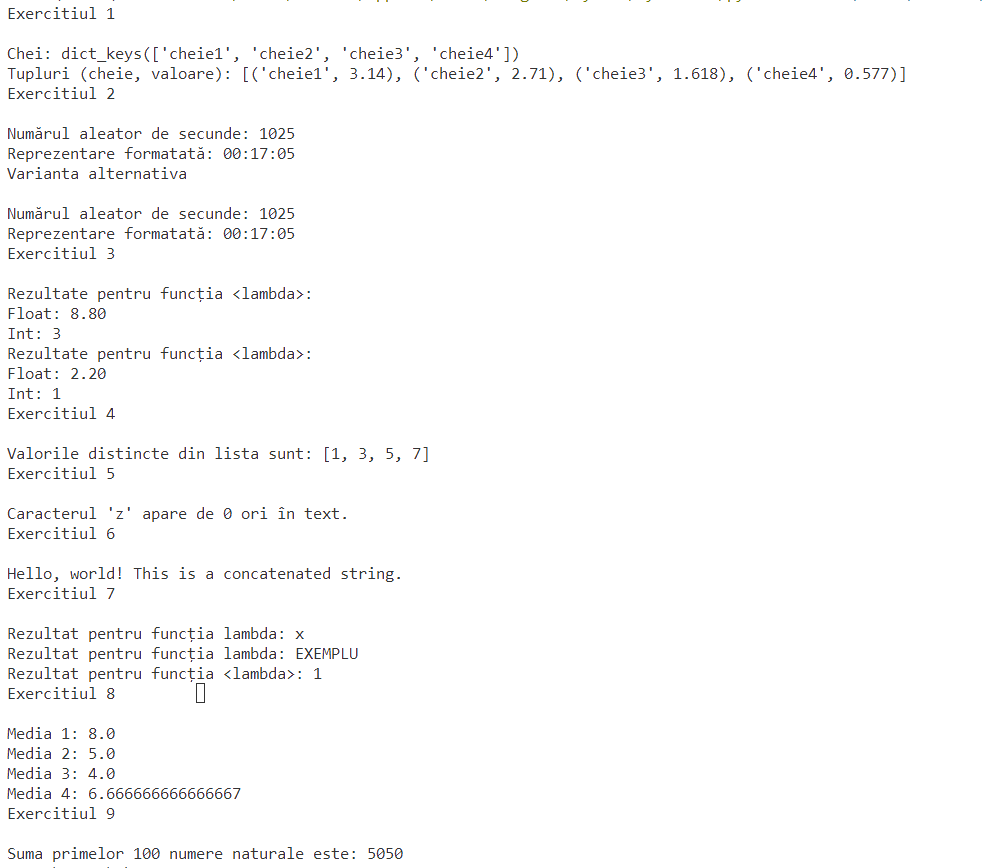
rezultat = suma\_primele\_N\_naturale(N)

print(f"Suma primelor {N} numere naturale este: {rezultat}")

- Aici se definește o funcție numită `suma\_primele\_N\_naturale` care primește un argument `N`.

- Funcția calculează suma primelor N numere naturale utilizând o abordare recursivă. Dacă N este mai mic sau egal cu 0, funcția returnează 0. Altfel, adaugă N la suma primelor N-1 numere naturale.

- Se exemplifică apelul funcției pentru suma primelor 100 de numere naturale și se afișează rezultatul.



**Concluzie**

Acest laborator a acoperit o varietate de concepte de programare în limbajul Python. Astfel am putut pune în practică aceste concepte, realizănd astfel urmpătoarele:

1. Am lucrat cu structuri de date precum dicționare, liste și șiruri de caractere. Am învățat cum să accesez și să manipulez elementele acestor structuri.

2. Am folosit funcții lambda pentru a crea funcții simple și anonime. Aceste funcții au fost apoi utilizate în contexte diferite, cum ar fi map(), filter() și reduce().

3. Am exersat manipularea șirurilor de caractere, inclusiv numărarea aparițiilor unui caracter dat și concatenarea a mai multor șiruri.

4. Am explorat modulul `random` pentru a genera numere aleatoare și am utilizat formatarea de șiruri pentru a prezenta rezultatele într-un format specific.

5. Am analizat situații în care se utilizează argumente implicite în funcții și am înțeles cum putem specifica anumite argumente în funcții folosind keyword arguments.

6. Am explorat recursivitatea prin calcularea sumei primelor N numere naturale.

Aceste exerciții m-au ajutat să exersez diverse abilități de programare și să dezvolt o mai bună înțelegere a limbajului Python. În plus, am dobândit experiență în rezolvarea unor probleme specifice utilizând concepte de programare.