Tema 8.
Funcții încorporate și decoratoare

Ce ne așteaptă?

1. Funcții recursive și funcții Lambda
2. Funcții încorporate în Python
3. Decoratoare în Python

1. Ce reprezintă o funcție recursivă?
2. Cum se definesc funcțiile Lambda în Python?
3. Care sunt funcțiile încorporate de bază în Python?
4. Care este rolul funcției enumerate()?
5. Care este rolul funcției map()?
6. Care este rolul funcției filtre()?
7. Care este rolul funcției reduce()?
8. Ce este un decorator în Python?
9. Cum se apelează o funcție cu extra-funcționalități?
10. Care este rolul simbolului @ în Python?

1. Funcții recursive și funcții Lambda

Funcții recursive

- **Funcția recursivă – funcția care se apelează pe ea însăși**
- **Utilitatea funcțiilor recursive:**

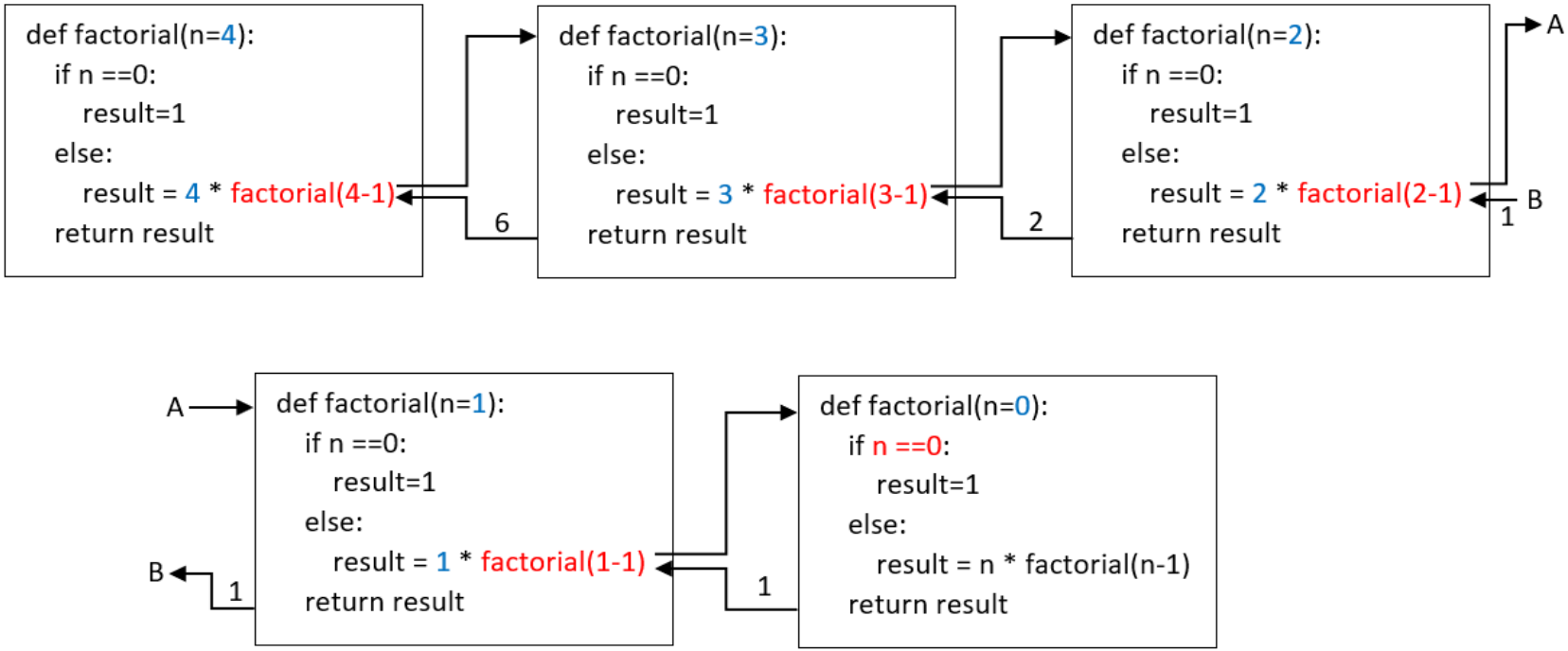
- reduce lungime codului și mentenanța lui
- reprezintă o cale simplă de soluționare a problemelor complexe

- **Exemplu funcției recursive – determinarea factorialului:**

```
def factorial(n):
    if n ==0:
        result=1
    else:
        result = n * factorial(n-1)
    return result

x= factorial(4)
print(f"Factorialul lui 4 este: {x}")
```

Funcția recursivă de determinare factorial



Funcții Lambda (anonime)

- **Funcția Lambda – definita într-o singura linie, fără nume, utilizează cuvantul-cheie *lambda***
- **Sintaxa:**

  lambda *lista_parametri*: *expresie*

- **Specificații:**
  - poate lua mai mulți parametri dar doar o singură expresie
  - efectuează operații asupra parametrilor și returnează rezultatul asemenea funcțiilor obișnuite
  - codul devine mai concis și ușor de citit
  - se utilizează doar în locul unor funcții simple și nu pentru operații complexe
  - de multe ori, se utilizează în combinație cu unele funcții: **map(), reduce(), filter() etc**

- **Exemple:**

```
s = lambda a: a*a
x=s(4)
print(x)

suma = lambda a,b: a+b
x=suma(4,5)
print(x)
```

2. Funcții încorporate în Python

https://www.w3schools.com/python/python_ref_functions.asp

Funcțiile de conversie *int(), float(), str(), list(), tuple(), set() și altele*

- ***Conversia unui tip de date în alt tip de date compatibil***
- **Sintaxa:**

  int(*obiect_compatibil*)

- **Exemplu de utilizare**

```
print(int(3.2))
print(float(5))
print(str(4.8))
print(list('3.2'))
print(tuple([3,6,2,8]))
print(set([3,4,7,3,5]))
```

Funcții de descriere a obiectelor

- type() – returnează tipul obiectului
- id() – returnează un numar de identificare a locație de memorie a obiectului
- len() – returnează lungimea obiectului
- sorted() – returnează o lista cu elementele sortate a obiectului
- print() – afișează valoarea obiectului la consolă
- input() – citește valoarea obiectului de la consolă
- Exemplu de utilizare

```
lista = [999, 888, 1100, 1200, 1300, 777]
print(lista)
print(type(lista))
print(id(lista))
print(len(lista))
print(sorted(lista))
```

Funcții operații cu valorile numerice

- abs() – returnează valoarea absolută
- max() – returnează valoarea maximă
- min() – returnează valoarea minimă
- pow() – returnează valoarea unui număr ridicat la o putere
- range() – returnează o secvență de numere întregi
- round() – rotunjește valoarea numărului la valoarea întreagă apropiată
- sum() – returnează suma elementelor unui obiect
- Exemplu de utilizare

```
num1, num2 = -2, 2.7
lista = [9, 8, 11, 12, 13, 7]
print(abs(num1))
print(max(lista))
print(min(lista))
print(pow(num2,num1))
print(round(num2))
print(sum(lista))
```

Funcția *enumerate()*

- **Returnează un obiect enumerate ce are ca elemente tupluri din câte două elemente. Primul element al tuplului este valoarea indexului, iar al doilea valoarea elementului secvenței atribuit funcției**
- **Sintaxa:**

  enumarate(*secvență*)

- **Exemple de utilizare**

```
lista = ["alb", "rosu", "verde"]
enum = enumerate(lista)
print(list(enum))

text="Pavel„
enum = enumerate(text)
tup = tuple(enum)
print(tup)
for i, j in tup:
    print(f'Litera {j} are indexul {i}'
```

Funcția *filter()*

- **Returnează un obiect filter ce conține elementele unei secvențe filtrate conform unei condiții**

- **Sintaxa:**

  filter(*funcție, secvență)*

  - parametrul funcție al funcție filter() returnează o condiție și se exprima prin funcție lambda

- **Exemplu de utilizare**

```
lista_valori = [999, 888, 1100, 1200, 1300, 777]
valori_satisf = filter(lambda x : x>1000, lista_valori)
lista_satisf=list(valori_satisf)
print(lista_valori)
print(lista_satisf)
```

Funcția *map()*

- **Returnează un obiect map ce conține toate elementele unei secvențe modificate după una și aceeași lege**

- **Sintaxa:**

  map(*funcție, secvență*)

  - parametrul funcție al funcție map() reprezintă legea de modificare și se exprima prin funcție lambda

- **Exemplu de utilizare**

```
lista_valori = [100, 200, 300, 400, 500, 600]
valori_modif = map(lambda x: x+27, lista_valori)
lista_modif=list(valori_modif)
print(lista_valori)
print(lista_modif)
```

Funcția *reduce()*

- **Reduce o secvență la un singur element utilizând o anumită logică**
- **Sintaxa:**

  reduce(*funcție, secvență*)

  - parametrul funcție al funcție reduce() reprezintă logica de reducere și se exprima prin funcție lambda

- **Exemplu de utilizare**

```
from functools import reduce
lista_valori = [100, 200, 300, 400, 500, 600]
valoare_final = reduce(lambda x, y: x+y, lista_valori)
print(lista_valori)
print(valoare_final)
```

3. Decoratoare în Python

Funcții decoratoare

- **Decoratorul – o funcție specială ce adaugă funcționalități suplimentare altei funcții**
- **Specificații:**

- decoratorul *decor* primește ca parametru o funcție *func* (căreia i se va adaugă extra funcționalități)
- decoratorul *decor* conține o funcție imbricata *func_im (*unde se specifică extra funcționalitățile)
- funcția imbricata *func_im* returnează funcția *func*
- decoratorul *decor* returnează funcția imbricată *func_im*

- **Sintaxa:**

```
def decor(func):
    def func_im(parametrii):
        corpul_functiei_imbricate
        return func(parametrii)
    return func_im
```

Exemplu decorator

- Funcția inițială – o funcție de adunare a 2 numere

```
def sumare(a,b):
    rezult = a + b
    return rezult
```

- Decoratorul - va permite sumarea doar a numerelor pozitive, cele negative egalându-le cu 0

```
def decor(func):
    def func_im(x,y):
        if x<0:
            x = 0
        if y<0:
            y = 0
        return func(x,y)
    return func_im
```

- Apelarea funcție de adunare a 2 numere cu extra funcționalități

```
adunare = decor(sumare)
print(adunare(20,30))
print(adunare(-10,5))
```

Simbolul @ în Python

- Simbolul @ urmat de denumirea decoratorul adaugă extra funcționalități funcției
- Exemplu de utilizare:

```
def decor(func):
    def func_im(x,y):
        if x<0:
            x = 0
        if y<0:
            y = 0
        return func(x,y)
    return func_im

@decor
def sumare(a,b):
    rezult = a + b
    return rezult

print(sumare(20,30))
print(sumare(-10,5))
```

Exemplu decorator în programarea web

```
...

@app.route("/")
def home():
   return render_template("./index.html")

@app.route("/job.html")
def job():
   return render_template("./job.html")

@app.route("/calcul.html")
def calcul():
   return
render_template("./calcul.html")

...
```