Curs2.Structuri repetitive,funcții matematice și aplicații în Python

Structuri repetitive

Bucla `for`

Folosită pentru a repeta instrucțiuni **de un număr cunoscut de ori** sau pentru a parcurge elementele unei mulțimi.

```
# parcurgere cu range
for i in range(5):
    print(i)
# → 0 1 2 3 4

# parcurgere listă(similar cu auto din C++)
fructe = ["mere", "pere", "banane"]
for f in fructe:
    print(f)
# → mere pere banane
```

Bucla `while`

```
# numărătoare până la 5
i = 1
while i <= 5:
    print(i)
    i += 1
# → 1 2 3 4 5

# citirea numerelor pana la întalnirea lui 0
while True:
    x = int(input("Număr (0 ca să ieși): "))
    if x == 0:
        break
    print("Ai introdus:", x)
```

Funcții matematice de bază

```
print(abs(-5))            # valoarea absolută → 5
print(round(3.14159, 2))  # rotunjire la 2 zecimale → 3.14
print(pow(2, 3))          # putere → 8 (similar cu 2**3)
```

```
x, y, z = 4, 7, 2
print(max(x, y, z))      # maximul dintre x, y, z → 7
print(min(x, y, z))      # minimul dintre x, y, z → 2
```

Modulul `math`

```
import math

print(math.sqrt(4))        # rădăcina pătrată → 2.0
print(math.pi)             # constanta π → 3.141592653589793
print(math.ceil(9.1))      # rotunjire în sus → 10
print(math.floor(9.6))     # rotunjire în jos → 9

print(math.factorial(5))   # factorial(5) → 120
print(math.log10(1000))    # logaritm în baza 10 → 3
print(math.log2(8))        # logaritm în baza 2 → 3
print(math.gcd(24, 36))    # cel mai mare divizor comun → 12
print(math.lcm(4, 6))      # cel mai mic multiplu comun → 12
```

Funcții trigonometrice

```
angle = math.radians(30)   # transformă grade în radiani
print(math.sin(angle))     # sin(30°) → 0.5
print(math.cos(angle))     # cos(30°) → 0.866...
print(math.tan(angle))     # tan(30°) → 0.577...

# Funcții inverse (în grade)
print(math.degrees(math.asin(0.5)))  # asin(0.5) → 30°
print(math.degrees(math.acos(0.5)))  # acos(0.5) → 60°
print(math.degrees(math.atan(1)))    # atan(1) → 45°
```

Funcții (subprograme)

- În Python, funcțiile sunt blocuri de cod reutilizabile care îți permit să grupezi instrucțiuni pentru a le apela ori de câte ori ai nevoie.
- Ele îți fac programele mai clare, mai scurte și mai ușor de întreținut.
- O funcție este un bloc de cod reutilizabil, definit cu `def`.
- (OBS: Poți returna ORICE fel de tip de date)

```
def nume_functie(parametri):
    # bloc de cod
    return rezultat
```

Exemple:

Factorial (iterativ)

```
def factorial(n):
    prod = 1
    for i in range(1, n+1):
        prod *= i
    return prod

print(factorial(5))  # → 120
```

Factorial (recursiv)

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

print(factorial(5))  # → 120
```

Descompunere în factori primi

```
def factori_primi(n):
    divizori = []
    d = 2
    while n > 1:
        while n % d == 0:
            divizori.append(d)
            n //= d
        d += 1
    return divizori

print(factori_primi(84))  # → [2, 2, 3, 7]
```

Descompunere în factori primi cu putere

```
def fact_primi_put(n):
    div = {}
    d = 2
    while n > 1:
        put = 0
        while n % d == 0:
            put += 1
```

```
            n //= d
        if put > 0:
            div[d] = put
        d += 1
    return div

print(fact_primi_put(84))  # → {2: 2, 3: 1, 7: 1}
```

`if __name__ == "__main__":`

- **Adevărat** → codul rulează direct din fișier.
- **Fals** → codul nu se execută automat la `import`.

Util pentru a separa **punctul de intrare** de restul codului.

Funcții lambda (funcții anonime)

Formă scurtă pentru funcții simple:

```
# definiție clasică
def suma(x, y):
    return x + y

# cu lambda
adunare = lambda x, y: x + y
```

Exemple cu `map` și `filter`:

```
lista = [2, 3, 4, 5, 3]

lista2 = list(map(lambda x: x*2, lista))    # → [4, 6, 8, 10, 6]
lista3 = list(filter(lambda x: x%2 == 0, lista))  # → [2, 4]
```

- **OBS**:
  - Funcția `map` oferă posibilitatea de a aplica o altă funcție asupra elementelor unei liste.(Returnează un obiect de tip `map`)
  - :Funcția `filter` aplică o filtrare după o anumită funcție a elementelor listei.

Aplicații finale (mini-exerciții)

1. Perimetrul cercului

```python
import math

def perimetru_cerc(r):
    return 2 * math.pi * r

print(round(perimetru_cerc(5), 2))  # → 31.42
```

---

2. Descompunere în factori primi (doar divizori)

```python
def factori_primi(n):
    divizori = []
    d = 2
    while n > 1:
        while n % d == 0:
            divizori.append(d)
            n //= d
        d += 1
    return set(divizori)  # doar divizorii, fără repetiții

print(factori_primi(84))  # → {2, 3, 7}
```