Tema 11.
Metodele claselor și moștenirea

Ce ne așteaptă?

1. Metode în clase Python
2. Moștenirea în Python
3. Ordinea accesării metodelor moștenite

1. Ce reprezintă o metodă a clasei?
2. Ce reprezintă o metodă de instanță și cum se definește ea?
3. Care este rolul metodelor setter și getter?
4. Ce reprezintă o metodă de clasă și cum se definește ea?
5. Ce reprezintă o metodă statică și cum se definește ea?
6. Ce reprezintă clasele imbricate?
7. Care este rolul moștenirii?
8. Ce tipuri de moștenire se definesc în Python?
9. Care este ordinea accesării metodelor moștenite?

1. Metode în clase Python

Tipuri de metode în clase

- Metodele din clase sunt funcții ce definesc comportamentele obiectelor

- Metode de instanță – se limitează la un singur obiect
- Metode de clasă – se limitează la toate obiectele clasei
- Metode statice – nu sunt legate de obiecte

Metode de instanță

- **Metodele de instanță - metode care acționează asupra variabilelor de instanță**
- **Primul parametru al metodelor de instanță este variabila *self***

```
class Om:
    def met(self, nume, inaltime):
        self.nume=nume
        self.inaltime=inaltime

persoana =Om()
persoana.met(„Elena", 147)
print(persoana.nume)
print(persoana.inaltime)
```

- **Metodele de instanță de obicei sunt metode:**

  - de tip setter
  - de tip getter

Metode setter

- Metodele setter se utilizează pentru setarea unui atribut al obiectului
- Metodele setter au drept al doilea parametru valoarea atributului setat
- Metodele setter nu returnează nimic
- Sintaxa metodelor setter:

def *set_variabila*(self, *variabila*):
self.*variabila* = *variabila*

```
class Om:
    def set_nume(self, nume):
        self.nume = nume
    def set_inaltime(self, inaltime):
        self.inaltime = inaltime

persoana = Om()
persoana.set_nume("Ana")
persoana.set_inaltime(150)
print(f'{persoana.nume} are inaltimea de {persoana.inaltime} cm')
```

Metode getter

- Metodele getter se utilizează pentru obținerea valorii unui atribut al obiectului
- Metodele getter au ca parametru doar variabila self
- Metodele getter returnează valoarea atributului respectiv
- Sintaxa metodelor getter

def *get_variabila*(self):
    return self.*variabila*

```
class Om:
    def __init__(self, nume, inaltime):
        self.nume = nume
        self.inaltime = inaltime
    def get_nume(self):
        return self.nume
    def get_inaltime(self):
        return self.inaltime

persoana = Om("Maria", 155)
n = persoana.get_nume()
i = persoana.get_inaltime()
print(f"{n} are inaltimea de {i} cm")
```

Metode de clasă

- Metodele de clasă - metode care acționează asupra variabilelor statice
- Metodele de clasă se declară cu decoratorul @classmethod
- Primul parametru al metodelor de clasă este variabila *cls*
- Metodele de clasă se utilizează rar în Python

```
class Om:
    num_maini=2

    @classmethod
    def lucreaza(cls):
        print(f"Persoana lucreaza utilizand {cls.num_maini} maini")

Om.lucreaza()
```

Metode statice

- Metodele statice – în general metode utilitare ce nu depind de nici un obiect
- Metodele statice se declară cu decoratorul @staticmethod
- Nu includ nici un prim parametru de tip variabile *self* sau *cls*
- De obicei se accesează utilizând numele clasei

```
class Om:
    @staticmethod
    def aduna(x,y):
        print(f"Suma dintre {x} si {y} este {x+y}")

    @staticmethod
    def imulteste(x,y):
        print(f"Produsul dintre {x} si {y} este {x*y}")

Om.aduna(2,3)
Om.imulteste(2,3)
```

Clase imbricate

- Clasă imbricată - o clasă în interiorul altei clase
- Obiectul clasei imbricate se va forma după instanțierea clasei externe
- În exteriorul claselor obiectul clasei imbricate se va forma folosind obiectul clasei externe

```
class Om:
    def __init__(self):
        print("Am creat o clasa Om")

    class Cap:
        def __init__(self):
            print("Am creat o clasa Cap in clasa Om")
        def gandeste(self):
            print("Omul gandeste cu capul")

persoana=Om()
cap_persoana = persoana.Cap()
cap_persoana.gandeste()
```

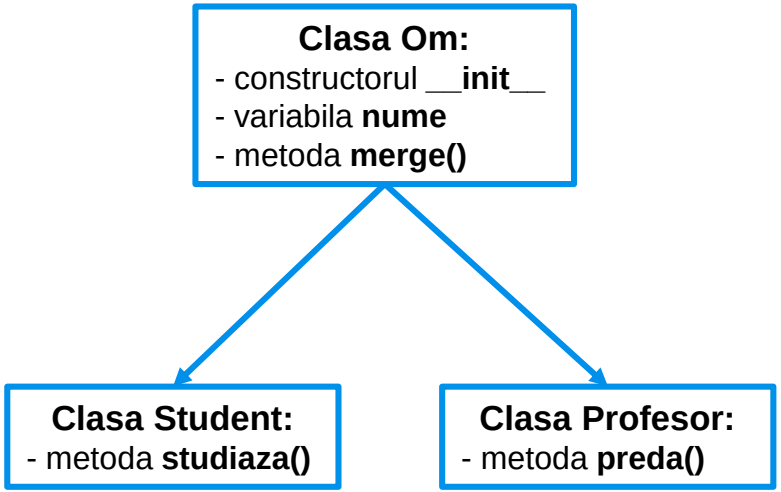
2. Moștenirea în Python

Noțiune de moștenire

- **Moștenire – trecerea atributelor și metodelor unei clase în altă clasă**
- **Procesul de moștenire presupune:**
  - Crearea unei clase noi în care se specifică ca parametru o clasă existentă
  - Clasa existentă se numește superclasă (clasă de bază sau clasă parinte)
  - Clasa creată se numește subclasă (clasă deviată sau clasă copil)
  - Variabilele, metodele și constructorii superclasei se moștenesc în subclasă
- **Avantajele moștenirii:**
  - Reutilizarea codului
  - Timp redus la elaborarea programului
  - Evitarea repetării codului

Exemplu de moștenire în Python

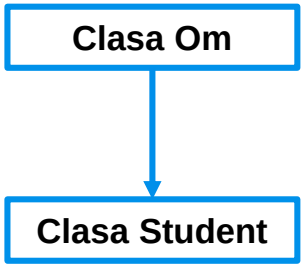
- La declararea subclasei, numele superclasei se trece ca parametru în paranteze rotunde



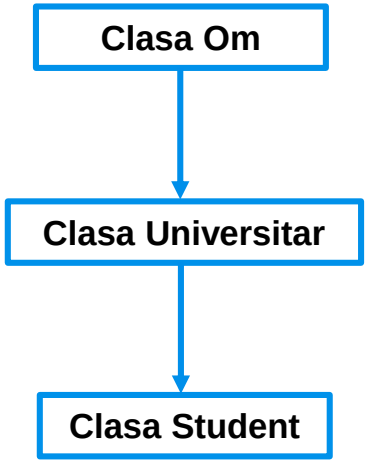

```
class Om:
    def __init__(self, nume):
        self.nume = nume
    def merge(self):
        print(f"{self.nume} este un om")

class Student(Om):
    def studiaza(self):
        print(f"{self.nume} este un student")

class Profesor(Om):
    def preda(self):
        print(f"{self.nume} este un profesor")

student = Student("Natalia")
print(student.nume)
student.merge()
student.studiaza()

profesor = Profesor("Olga")
print(profesor.nume)
profesor.merge()
profesor.preda()
```

Tipuri de moștenire

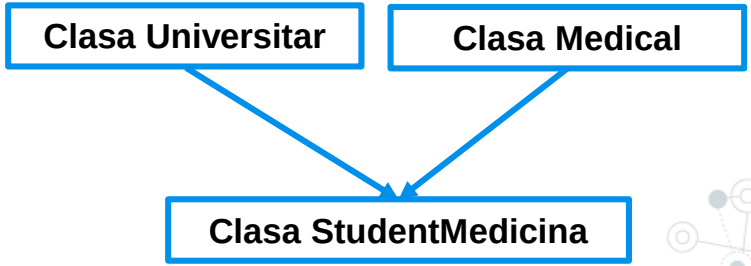
- Moștenire singulară



- Moștenire multinivel



- Moștenire multiplă



Moștenire singulară

- **Subclasa are o singură superclasă (poate avea mai multe clase “surori”)**

```
class Om:
    def __init__(self, nume):
        self.nume = nume
    def merge(self):
        print(f"{self.nume} este un om")

class Student(Om):
    def studiaza(self):
        print(f"{self.nume} este un student")

student = Student(„Alexandra")
print(student.nume)
student.merge()
student.studiaza()
```

Moștenirea multinivel

- Subclasa are o singură superclasă dar aceasta este subclasa altei superclase

```
class Om:
    def __init__(self, nume):
        self.nume = nume
    def merge(self):
        print(f"{self.nume} este un om")

class Universitar(Om):
    def cerceteaza(self):
        print(f"{self.nume} este un universitar")

class Student(Universitar):
    def studiaza(self):
        print(f"{self.nume} este un student")

student = Student("Veronica")
print(student.nume)
student.merge()
student.cerceteaza()
student.studiaza()
```

Moștenire multiplă

- Subclasa are mai multe superclase

```
class Universitar:
    def __init__(self, nume):
        self.nume=nume
    def cerceteaza(self):
        print(f"{self.nume} este un universitar")

class Medical:
    def __init__(self, nume):
        self.nume=nume
    def injecteaza(self):
        print(f"{self.nume} este un medical")

class Student(Universitar, Medical):
    def studiaza(self):
        print(f"{self.nume} este un student")

student = Student("Daniela")
print(student.nume)
student.cerceteaza()
student.injecteaza()
student.studiaza()
```

Superclase cu metode cu același nume

- Se moștenește metoda primei superclase specificate în paranteze rotunde

```
class Universitar:
    def __init__(self, nume):
        self.nume=nume
    def publica(self):
        print(f"{self.nume} este un universitar")

class Medical:
    def __init__(self, nume):
        self.nume=nume
    def publica(self):
        print(f"{self.nume} este un medical")

class Student(Universitar, Medical):
    def studiaza(self):
        print(f"{self.nume} este un student")

student = Student(„Mariana")
print(student.nume)
student.publica()
student.studiaza()
```

Moștenirea constructorului

- Constructorul superclasei se moștenește automat în subclasă
- Dacă subclasa are constructor atunci acesta are prioritate față de cel din superclasă
- Pentru apelarea constructorului din superclasa se utilizează funcția super()

```
class Om:
    def __init__(self):
        print("Constructorul superclasei")

class Student(Om):
    def __init__(self):
        print("Constructorul subclasei")
        #super().__init__()

student = Student()
```

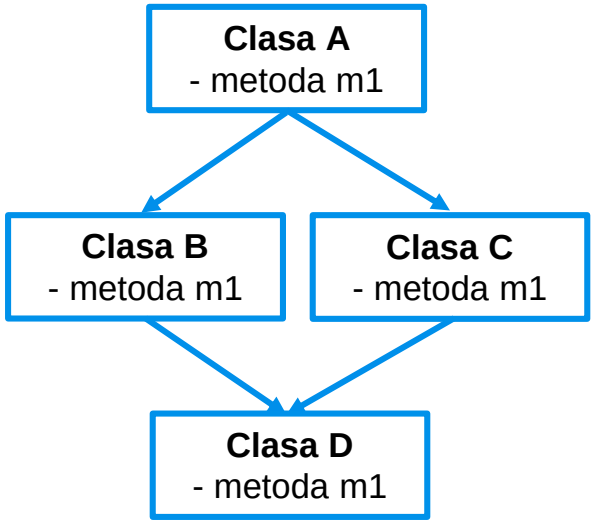
3. Ordinea accesării metodelor moștenite

Principiile accesării metodelor moștenite

- **Method Resolution Order (MRO)**
- **Principiile accesării metodelor claselor în ierarhia de moștenire**
  - Inițial metoda apelată se caută în subclasă
  - Dacă metoda apelată nu se află în subclasă atunci se caută în superclase în ordinea declarării lor la moștenire
  - Odată ce metoda a fost căutată într-o clasă atunci la această clasă nu se va mai reveni
- **Metoda mro() – permite vizualizarea ordinii executării claselor**
- **Sintaxa metodei mro():**

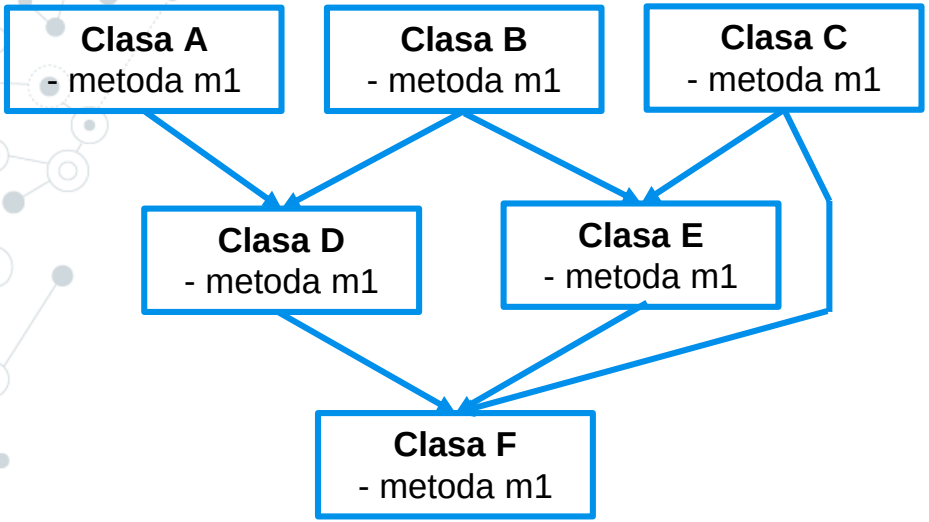
***NumeClasa*.mro()**

Exemplu de accesare a metodelor moștenite (1)




```
class A:
    def m1(self):
        print("m1 din clasa A")
class B(A):
    def m1(self):
        print("m1 din clasa B")
class C(A):
    def m1(self):
        print("m1 din clasa C")
class D(B, C):
    def m1(self):
        print("m1 din clasa D")

d=D()
d.m1()
print(A.mro())
print(B.mro())
print(C.mro())
print(D.mro())
```

Exemplu de accesare a metodelor moștenite (2)




```
class A:
    def m1(self):
        print("m1 din A")
class B:
    def m1(self):
        print("m1 din B")
class C:
    def m1(self):
        print("m1 din C")
class D(A, B):
    def m1(self):
        print("m1 din D")
class E(B, C):
    def m1(self):
        print("m1 din E")
class F(D, E, C):
    def m1(self):
        print("m1 din F")
print(A.mro())
print(B.mro())
print(C.mro())
print(D.mro())
print(E.mro())
print(F.mro())
```