Tema 12. Polimorfism, clase abstracte și interfețe

Ce ne așteaptă?

1. Funcția *super()* în Python
2. Polimorfismul în Python
3. Clase abstracte
4. Interfețe în Python

1. Care este rolul funcției *super()*?
2. Ce reprezintă polimorfismul?
3. Ce reprezintă supraîncărcarea și supraînscrierea?
4. Care este esența supraîncărcarii operatorilor?
5. Care este esența supraîncărcarii metodelor și a constructorului?
6. Care este esența supraînscrierii metodelor și a constructorului?
7. Ce reprezintă și cum se definesc clasele abstracte?
8. Care este rolul și cum se definesc metodele abstracte?
9. Ce reprezintă și cum se definesc interfețele?

1. Funcția *super()* în Python

Utilizarea funcției *super()*

- **Funcția super() – permite accesarea din cadrul subclasei a:**
  - Constructorului superclasei
  - Variabilelor superclasei
  - Metodelor superclasei

- **Se utilizează când metoda sau variabila au același nume în subclasă și superclasă**
- **Sintaxa apelării metodelor din superclasa:**
  - Fără utilizarea funcției super():

    ***NumeSuperClasa*.*nume_metoda*(self)**
  - Cu utilizarea funcției super():

    super(*NumeSuperClasa*, self).*nume_metoda*

Exemplu de apelare a constructorului parametrizat cu funcția *super()*

```
class Om:
    def __init__(self, nume, prenume):
        self.nume=nume
        self.prenume=prenume
    def afisare(self):
        print(f"Aceasta persoana se numeste {self.nume} {self.prenume}")

class Student(Om):
    def)
        self.grupa=grupa__init__(self, nume, prenume, grupa):
        super().__init__(nume, prenume
    def afisare(self):
        print(f"Aceast student se numeste {self.nume} {self.prenume} si este din
grupa {self.grupa}")

persoana = Om("Nicolaev", "Pavel")
persoana.afisare()

student = Student("Nicolaev", "Pavel", "TLC-091")
student.afisare()
```

Exemplu de apelare a variabilelor și metodelor cu funcția *super()*

- **Apelarea variabilei superclasei cu același nume**

```
class A:
    x=10
class B(A):
    x=20
    def m1(self):
        print(f'Variabila x in subclasa are valoarea {self.x}’)
        print(f'Variabila x in superclasa are valoarea {super().x}')
b=B()
b.m1()
```

- **Apelarea metodei superclasei cu același nume**

```
class A:
    def m1(self):
        print("Metoda m1 din superclasa A")
class B(A):
    def m1(self):
        print("Metoda m1 din subclasa B")
        super().m1()
b=B()
b.m1()
```

2. Polimorfismul în Python

Tipuri de polimorfism

- **Supraîncărcarea – utilizarea același metode sau operator pentru diferite scopuri**
  - Supraîncărcarea metodelor – nu este susținută în Python
  - Supraîncărcarea operatorilor
  - Supraîncărcarea constructorului – nu este susținută în Python
- **Suprascrierea – redefinirea în subclasă a metodei sau a operatorului moștenit**
  - Suprascrierea metodelor
  - Suprascrierea constructorului

Supraîncărcarea operatorilor

- **Operatorul + pentru sumarea numerelor sau concatenarea stringurilor/listelor**

```
print(10+20)
print(„Limbajul " + "Python")
print([1,2,3]+[4,5,6])
```

- **Operatorul * pentru înmulțirea numerelor sau multiplicarea stringurilor/listelor**

```
print(10*20)
print("Python "*3)
print([1,2,3]*3
```

Supraîncărcarea metodelor și constructorului – nesusținut de Python

- **Exemplu de supraîncărcare a metodelor**

```
class A:
    def m1(self):
        print('Metoda m1 fara parametri’)
    def m1(self, a):
        print('Metoda m1 cu un parametru’)
    def m1(self, a, b):
        print('Metoda m1 cu 2 parametri’)
a=A()
a.m1()
```

- **Exemplu de supraîncărcare a constructorului**

```
class B:
    def __init__(self):
        print('Constructor fara parametri’)
    def __init__(self, a):
        print(‘Constructor cu un parametru’)
    def __init__(self, a, b):
        print('Constructor cu 2 parametri’)

b=B()
```

Suprascrierea metodelor

- **Suprascrierea metodei – redefinirea în subclasă a metodei moștenite din superclasa**

```
class Om:
    def primeste_bani(self):
        print('Merge cu cardul la bancomat')

class Student(Om):
    def primeste_bani(self):
        print("Daca are rezultate bune, are bursa si")
        print('merge cu cardul la bancomat')

class Profesor(Om):
    def primeste_bani(self):
        print("Daca are lectii si preda atunci")
        print('merge cu cardul la bancomat')

persoana=Om()
persoana.primeste_bani()
student=Student()
student.primeste_bani()
profesor=Profesor()
profesor.primeste_bani()
```

Suprascrierea metodei __str__()

- Metoda __str__() permite concretizarea modului de afișare a obiectelor
- Exemplu fără metoda __str__()

```
class Student:
    def __init__(self, numeprenume, grupa):
        self.numeprenume=numeprenume
        self.grupa=grupa
s1=Student('Mihai Eminescu', "TLC-151")
s2=Student('Ion Creanga' , "TLC-164")
print(s1)
print(s2)
```

- Exemplu cu metoda __str__()

```
class Student:
    def __init__(self, numeprenume, grupa):
        self.numeprenume=numeprenume
        self.grupa=grupa
    def __str__(self):
        return f"Studentul {self.numeprenume} este din grupa {self.grupa}"
s1=Student('Mihai Eminescu', "TLC-051")
s2=Student('Ion Creanga' , "TLC-064")
print(s1)
print(s2)
```

Suprascrierea constructorului (1)

- Dacă în subclasă lipsește constructorul, se moștenește constructorul superclasei

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

class Student(Om):
    def afisare(self):
        print(f"Numele studentului este {self.nume}")

student1=Student("Irina")
student1.afisare()
student2=Student()
```

Suprascrierea constructorului (2)

- Dacă în subclasă este prezent constructorul, constructorul superclasei se ignoră

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

class Student(Om):
    def __init__(self, grupa):
        self.grupa=grupa

student=Student("TLC-121")
print(student.grupa)
print(student.nume)
```

Suprascrierea constructorului (3)

- **Dacă în subclasă este prezent constructorul, constructorul superclasei se apelează cu super()**

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

class Student(Om):
    def __init__(self, nume, grupa):
        super().__init__(nume)
        self.grupa=grupa

student=Student("Ina", "TLC-133")
print(student.grupa)
print(student.nume)
```

3. Clase abstracte

Definiția clasei abstracte

- Clasa abstractă – clasa care are cel puțin o metodă abstractă
- Metoda abstractă (nedefinită, non-concretă) – metoda care are doar nume nu și corp
- Clasa abstractă în Python obligatoriu va moșteni clasa *ABC* din modulul *abc* importat
- Clasa abstractă nu poate fi instanțiată adică nu se pot forma obiecte ale acestei clase
- Metoda abstractă se va declara cu ajutorul decoratorului *@abstractmethod*
- Metoda abstractă va fi implementată de subclasa a clasei abstracte
- Dacă subclasa clasei abstracte nu implementează metoda aceasta devine și ea abstractă
- Clasa abstractă poate conține și variabile, constructor și metode neabstacte

Metode abstracte

- **Metoda abstractă are doar nume nu și corp**
- **Metoda abstractă se declară cu decoratorul *@abstractmethod***
- **Pentru utilizarea decoratorului *@abstractmethod* este necesar importul modulului *abc***
- **În calitate de corp al metodei abstracte se utilizează cuvântul cheie *pass***
- **Metoda abstractă se va implementa de toate clasele neabstracte ce moștenesc clasa abstractă**
- **Sintaxa metodei abstracte:**

```
@abstractmethod
def nume_metoda(self):
    pass
```

Exemplu de clasă abstractă

```
from abc import *
class Om(ABC):
    def __init__(self, nume):
        self.nume=nume
    @abstractmethod
    def vorbeste(self):
        pass
    def gandeste(self):
        print("Metoda neabstracta in clasa abstracta Om")
class Universitar(Om):
    def __init__(self, nume, prenume):
        super().__init__(nume)
        self.prenume=prenume
    def cerceteaza(self):
        print("Metoda neabstracta in clasa abstracta Universitar")
class Student(Universitar):
    def __init__(self, nume, prenume, grupa):
        super().__init__(nume, prenume)
        self.grupa=grupa
    def vorbeste(self):
        print("Metoda abstracta implementata in clasa Student")
student=Student("Popescu", "Adrian", "TLC-171")
student.vorbeste()
student.gandeste()
student.cerceteaza()
#persoana=Om("Popescu")
#universitar=Universitar("Popescu", "Adrian")
```

Subclase ale clasei abstracte

- O clasă abstractă poate avea mai multe subsclase directe
- Fiecare subclasă a clasei abstracte va trebui să implementeze metodele abstracte pentru a nu deveni și ele abstracte

```
from abc import *
class Om(ABC):
    def __init__(self, nume, prenume):
        self.nume=nume
        self.prenume=prenume
    @abstractmethod
    def vorbeste(self):
        pass
    def gandeste(self):
        print("Metoda gandeste ca un Om")

class Student(Om):
    def vorbeste(self):
        print("Metoda vorbeste studentul")
class Profesor(Om):
    def vorbeste(self):
        print("Metoda vorbeste profesorul")
student=Student("Popov", "Iurie")
student.vorbeste()
student.gandeste()
profesor=Profesor("Ionecu", "Eugen")
profesor.vorbeste()
profesor.gandeste()
```

Moștenirea clasei ABC

- Dacă clasa are metodă abstractă ar nu moștenește clasa ABC atunci aceasta poate fi instanțiată
- Metoda abstractă se va considera o metodă “pustie” fără moștenirea clasei ABC

```
from abc import *

class Om:
    @abstractmethod
    def vorbeste(self):
        pass

    def gandeste(self):
        print("Metoda neabstracta")

persoana=Om()
persoana.vorbeste()
persoana.gandeste()
```

4. Interfețe în Python

Definiția Intefeței

- **Interfața – o clasă abstractă ce conține doar metode abstracte**
- **O interfață poate conține și constructor și variabile însă doar metode abstracte**
- **Metodele abstracte ale interfeței vor fi implementate în subclasele acesteia**
- **O interfață nu poate fi instanțiată, adică nu se poate crea un obiect al acesteia**
- **O interfață moștenește clasa ABC**
- **O interfață se declară exacta ca o clasă abstractă respectându-se cerințe specifice**
  - În alte limbaje, de exemplu Java, există cuvinte cheie pentru definirea interfeței, de exemplu Inteface

Exemplu de interfață

```
from abc import *
class Om(ABC):
    @abstractmethod
    def vorbeste(self):
        pass
    @abstractmethod
    def gandeste(self):
        pass
class Student(Om):
    def vorbeste(self):
        print("Metoda vorbeste implementata in clasa Student")
    def gandeste(self):
        print("Metoda gandeste implementata in clasa Student")
class Profesor(Om):
    def vorbeste(self):
        print("Metoda vorbeste implementata in clasa Profesor")
    def gandeste(self):
        print("Metoda gandeste implementata in clasa Profesor")

student=Student()
student.vorbeste()
student.gandeste()

profesor=Profesor()
profesor.vorbeste()
profesor.gandeste()
```