OBJEKTUETARA BIDERATUTAKO PROGRAMAZIOA

Web garapena zerbitzari ingurunean

Otaola Hiribidea, 29
20600 Eibar, Gipuzkoa
T_943 89 92 11
idazkaritza@uni.eus
www.uni.eus
@uni.eus

UNI EIBAR ERMUA

EUSKO JAURLARITZA
GOBIERNO VASCO
HEZKUNTZA SAILA
DEPARTAMENTO DE EDUCACIÓN

Aurkibidea

1. OBJEKTUETARA BIDERATUTAKO PROGRAMAZIOA (OOP)

Programazio modelo honen oinarria objektuak dira, beraz gure programa diseinatzeko orduan arazoa eta soluzioaren parte diren objektuetan zentratuko gara ez funtzioetan.

Objektua atributu eta izaera propioa daukan datu multzoa bezala definitu dezakegu.

Hurrengo lerroetan objektuen sorrera erabileraren inguruan arituko gara:

```
class Gelatina:
    def __init__(self, tam, kolore, zapore):
        self.tam = tam
        self.kolore = kolore
        self.zapore = zapore

    def inprimatu(self):
         print(f"Zure gelatinaren tamaina {self.tam}, kolorea {self.kolore}
eta zaporea {self.zapore}")
```

Lehenengo lerroan objektuaren izena ezarriko dugu, __init__ funtzioak bere ezaugarriak definitzen ditu eta sortzeko aukera ematen digu.

Direktorio berdinean dagoen beste .py fitxategi batetatik erabili ahalko dugu hurrengoa eginez:

```
import Gelatina

gel1 = Gelatina.Gelatina("handia", "gorria", "marrubia")
gel1.inprimatu()
```

Horrela gelatina bat sortuko duen programa egiten dugu eta hurrengoa aterako da kontsolatik:

```
Zure gelatinaren tamaina handia, kolorea gorria eta zaporea marrubia
```

2. HERENTZIA

Kasu batzuetan objektuek ezaugarri amankomunak dituzte, hori kudeatzeko herentzia erabiliko dugu. Herentziak ezaugarri amankomunak superklase batean gordeko ditu eta objektu bakoitzaren ezaugarri bereziak objektuak kudeatuko ditu:

```
class Etxetresna:
    def __init__(self, izena):
        self.izena = izena
        self.piztuta = False

    def piztu(self):
        if(self.piztuta):
            print("Etxetresna jada piztuta zegoen.")
        else:
            self.piztuta = True
            print("Etxetresna piztu duzu.")

    def itzali(self):
        if(self.piztuta):
            self.piztuta = False
            print("Etxetresna itzali duzu.")
        else:
            print("Etxetresna jada itzalita zegoen.")

class Mugikorra(Etxetresna):
    def __init__(self, izena, sistema_eragilea):
        super().__init__(izena)
        self.sistema_eragilea = sistema_eragilea

    def getSistemaEragilea(self):
        print(f"Mugikorraren sistema eragilea {self.sistema_eragilea} da.")

class Garbigailua(Etxetresna):
    def __init__(self, izena, garbitze_tenperaturak):
        super().__init__(izena)
        self.garbitze_tenperaturak = garbitze_tenperaturak
```

```
    def getGarbitzeTenperaturak(self):
         print(f"Garbitzailearen garbitze-tenperaturak
{self.garbitze_tenperaturak} °C dira.")
```

Ondorioz karpeta berdinean dagoen beste script batean hurrengoa egin dezakegu:

```
mug= Etxetresnak.Mugikorra("Iphone 7","IOS")
mug.piztu()

garb=Etxetresnak.Garbigailua("LG",[40, 70])
garb.getGarbitzeTenperaturak()
```

Eta emaitza hurrengoa izango da:

```
Etxetresna piztu duzu.
Garbitzailearen garbitze-tenperaturak [40, 70] °C dira.
```

2.1 HERENTZIA ANITZA

Klase batek bi superklase izan ditzake Python-en, hau da, bi klase ezberdinen ezaugarriak jaso ditzake. Adibidez, gure aurreko adibidea oinarri bezala hartuta agian gure arazoa konpontzeko telefonoa izeneko objektua beharko dugu.

Ondorioz, mugikorrak etxetresnen ezaugarriak eta telefono objektuaren ezaugarriak erabiliko ditu:

```
class Telefonoa:
    def deitu(self):
        print("Deitzen...")
    def eskegi(self):
        print("Telefonoa eskegi da.")

class Mugikorra(Etxetresna, Telefonoa):
    def __init__(self, izena, sistema_eragilea):
        super().__init__(izena)
        self.sistema_eragilea = sistema_eragilea

    def getSistemaEragilea(self):
        print(f"Mugikorraren sistema eragilea {self.sistema_eragilea} da.")
```

Ondorioz hurrengoa egiten badugu:

```
mug= Etxetresnak.Mugikorra("Iphone 7","IOS")
mug.piztu()
mug.deitu()
```

Hurrengo emaitza lortuko dugu:

```
Etxetresna piztu duzu.
Deitzen...
```

Etxetresna eta Telefono klaseek izen bereko metodoren bat izango balute, ezkerreko oinordetzak lehentasuna du, kasu honetan Etxetresna klaseko metodoa hartuko luke.

3. KAPSULATZEA

Kapsulatzea, kanpotik klasearen barneko atributuetara era zuzenean sartzeko aukera ukatzean datza. Beste modu batean azaldu behar badugu gure klasea sarbide maila ezberdinetan bereiztea da.

Horretarako konstruktoreari (Python-en __init__) ez dizkiogu atributu ezberdinen balioak gehitzeko aukera emango, ez baititugu parametro bezala gehituko. Atributu bakoitzaren balioa aldatu eta ezagutzeko metodo bereziak erabiliko ditugu (getter/setter).

Aurreko adibidea aldatuz:

```
class Etxetresna:
    def __init__(self):
        self.piztuta = False

    def setIzena(self, izena):
        self.izena = izena

    def getIzena(self):
        return self.izena
```

```
    def piztu(self):
        if(self.piztuta):
            print("Etxetresna jada piztuta zegoen.")
        else:
            self.piztuta = True
            print("Etxetresna piztu duzu.")

    def itzali(self):
        if(self.piztuta):
            self.piztuta = False
            print("Etxetresna itzali duzu.")
        else:
            print("Etxetresna jada itzalita zegoen.")

class Garbigailua(Etxetresna):
    def __init__(self, izena, garbitze_tenperaturak):
        super().__init__()
        self.setIzena(izena)
        self.garbitze_tenperaturak = garbitze_tenperaturak

    def getGarbitzeTenperaturak(self):
         print(f"Garbitzailearen garbitze-tenperaturak
{self.garbitze_tenperaturak}  °C dira.")
```