**RECURSOS**

**APUNTS POO EN PYTHON**

**RA1: Genera diagrames de classes**:

* Identifica els conceptes bàsics de la programació orientada a objectes (1.1).
* Traça diagrames de classes a partir de les especificacions d'aquestes classes (1.5).
* Genera codi a partir d'un diagrama de classes (1.6).
* Genera un diagrama de classes mitjançant enginyeria inversa (1.7).

**RA2: Genera diagrames de comportament**:

* Identifica els diferents tipus de diagrames de comportament i el seu significat (2.1).
* Reconeix i elabora diagrames d'interacció, d'activitats i d'estats (2.4, 2.6, 2.8).

**1. Introducció a la Programació Orientada a Objectes**

La Programació Orientada a Objectes (POO) és un paradigma de programació basat en la creació d'objectes, que són instàncies de classes. En aquest paradigma, les dades i les funcions que operen sobre aquestes dades s'agrupen en entitats anomenades "objectes".

**2. Conceptes Fonamentals de la POO**

1. **Classe**: És un plànol o model que defineix els atributs (propietats) i mètodes (comportaments) que tindran els objectes.
2. **Objecte**: És una instància d'una classe. Un objecte té les propietats i comportaments definits per la classe.
3. **Mètode**: És una funció definida dins d'una classe, que descriu un comportament que els objectes d'aquesta classe poden tenir.
4. **Atribut**: Són les variables associades a una classe o objecte, que emmagatzemen dades sobre l'estat de l'objecte.
5. **Encapsulació**: És el mecanisme que permet agrupar dades i mètodes dins d'una classe i protegir-les de l'accés extern no controlat.
6. **Herència**: És el mecanisme que permet que una classe pugui heretar atributs i mètodes d'una altra classe.
7. **Polimorfisme**: És la capacitat d'utilitzar el mateix nom de mètode en diferents classes amb comportaments diferents.

**3. Creació de Classes i Objectes en Python**

Per crear una classe en Python, s'utilitza la paraula clau class. Un cop creada la classe, podem instanciar objectes d'aquesta classe.

**Exemple 1: Creació d'una Classe i Objecte**

# Definim una classe anomenada 'Cotxe'

class Cotxe:

# Mètode inicialitzador (constructor)

def \_\_init\_\_(self, marca, model):

self.marca = marca

self.model = model

# Un mètode per mostrar la informació del cotxe

def mostra\_info(self):

print(f"Marca: {self.marca}, Model: {self.model}")

# Creem un objecte de la classe Cotxe

cotxe1 = Cotxe("Toyota", "Corolla")

# Cridem el mètode mostra\_info

cotxe1.mostra\_info()

**Sortida:**

Marca: Toyota, Model: Corolla

**4. Encapsulació**

L'encapsulació permet ocultar la informació interna dels objectes i només exposar allò necessari mitjançant mètodes públics.

**Exemple 2: Encapsulació**

class Persona:

def \_\_init\_\_(self, nom, edat):

self.\_\_nom = nom # Atribut privat

self.\_\_edat = edat # Atribut privat

# Mètode públic per accedir a l'atribut nom

def obtenir\_nom(self):

return self.\_\_nom

# Mètode públic per modificar l'edat

def canviar\_edat(self, nova\_edat):

if nova\_edat > 0:

self.\_\_edat = nova\_edat

else:

print("L'edat ha de ser un valor positiu.")

# Crear una instància de la classe Persona

persona1 = Persona("Anna", 25)

# Intentar accedir directament als atributs privats (això produirà un error)

# print(persona1.\_\_nom) # Això provocarà un error

# Utilitzem els mètodes públics per accedir i modificar les dades encapsulades

print(persona1.obtenir\_nom()) # Anna

persona1.canviar\_edat(30)

**5. Herència**

L'herència permet crear una nova classe basada en una classe existent. La nova classe (classe filla) hereta els atributs i mètodes de la classe original (classe pare).

**Exemple 3: Herència**

# Definim la classe pare

class Animal:

def \_\_init\_\_(self, nom):

self.nom = nom

def parla(self):

print(f"{self.nom} està fent un so.")

# Definim una classe filla que hereta de la classe Animal

class Gos(Animal):

def parla(self):

print(f"{self.nom} està lladrant.")

# Creem un objecte de la classe filla

gos1 = Gos("Rex")

gos1.parla() # Rex està lladrant

**Sortida:**

Rex està lladrant.

**6. Polimorfisme**

El polimorfisme permet que diferents classes tinguin mètodes amb el mateix nom, però amb comportaments diferents.

**Exemple 4: Polimorfisme**

class Gat(Animal):

def parla(self):

print(f"{self.nom} està miolant.")

# Creem una llista d'animals

animals = [Gos("Rex"), Gat("Misu")]

# Fem que tots els animals parlin (polimorfisme)

for animal in animals:

animal.parla()

**Sortida:**

Rex està lladrant.

Misu està miolant.

Exercici 1

Ampliació de la Classe Cotxe amb Herència i Polimorfisme

Objectiu de l'activitat: L'objectiu d'aquesta activitat és posar en pràctica conceptes de programació orientada a objectes (POO), com la creació de classes i objectes, l'herència, l'encapsulació i el polimorfisme. A més, treballareu amb mètodes especials (com el constructor) i mètodes que gestionen l'estat dels objectes.

Desenvolupament de la tasca

1. Crear una classe Cotxe:
   * Afegeix els següents atributs:
     + marca: Cadena de text que representa la marca del cotxe.
     + model: Cadena de text que representa el model del cotxe.
     + velocitat\_max: Enter que representa la velocitat màxima del cotxe (en km/h).
     + \_\_encès: Atribut privat (bool) que indica si el cotxe està encès o apagat (per defecte, apagat).
   * Afegeix els següents mètodes:
     + encendre(): Aquest mètode ha d'actualitzar l'estat de l'atribut \_\_encès a True i mostrar un missatge indicant que el cotxe està encès.
     + apagar(): Aquest mètode ha de canviar l'estat de \_\_encès a False i mostrar un missatge indicant que el cotxe està apagat.
     + mostra\_info(): Aquest mètode ha de mostrar la marca, el model, la velocitat màxima i si el cotxe està encès o apagat.
2. Crear una subclasse CotxeElectric:
   * Aquesta subclasse ha d'heretar de Cotxe i afegir el següent atribut:
     + capacitat\_bateria: Enter que indica la capacitat de la bateria (en kWh).
   * Afegeix els següents mètodes a CotxeElectric:
     + carregar\_bateria(): Aquest mètode ha de mostrar un missatge indicant que la bateria del cotxe s'està carregant.
     + Sobreescriu el mètode mostra\_info() per afegir informació sobre la capacitat de la bateria, a més de la informació bàsica del cotxe.
3. Prova el teu programa:
   * Crea un objecte de la classe Cotxe i un altre de la classe CotxeElectric.
   * Utilitza els mètodes encendre(), apagar() i mostra\_info() per veure el comportament de cada objecte.
   * En el cas del cotxe elèctric, fes una crida addicional al mètode carregar\_bateria().

Requisits:

* La classe Cotxe ha de tenir l'atribut \_\_encès encapsulat com a atribut privat.
* La subclasse CotxeElectric ha de fer ús d'herència i polimorfisme.
* Implementar el mètode mostra\_info() de forma que sigui diferent per CotxeElectric.

# Classe base Cotxe

class Cotxe:

def \_\_init\_\_(self, marca, model, velocitat\_max):

self.marca = marca

self.model = model

self.velocitat\_max = velocitat\_max

self.\_\_encès = False # Atribut privat per indicar si el cotxe està encès o apagat

# Mètode per encendre el cotxe

def encendre(self):

if not self.\_\_encès:

self.\_\_encès = True

print(f"El cotxe {self.marca} {self.model} està encès.")

else:

print(f"El cotxe ja està encès.")

# Mètode per apagar el cotxe

def apagar(self):

if self.\_\_encès:

self.\_\_encès = False

print(f"El cotxe {self.marca} {self.model} està apagat.")

else:

print(f"El cotxe ja està apagat.")

# Mètode per mostrar informació del cotxe

def mostra\_info(self):

estat = "encès" if self.\_\_encès else "apagat"

print(f"Marca: {self.marca}, Model: {self.model}, Velocitat màxima: {self.velocitat\_max} km/h, Estat: {estat}")

# Subclasse CotxeElectric que hereta de Cotxe

class CotxeElectric(Cotxe):

def \_\_init\_\_(self, marca, model, velocitat\_max, capacitat\_bateria):

super().\_\_init\_\_(marca, model, velocitat\_max) # Iniciem els atributs de la classe pare

self.capacitat\_bateria = capacitat\_bateria # Nou atribut per la bateria (en kWh)

# Mètode per carregar la bateria

def carregar\_bateria(self):

print(f"La bateria del {self.marca} {self.model} s'està carregant.")

# Sobreescrivim el mètode mostra\_info per afegir informació sobre la bateria

def mostra\_info(self):

super().mostra\_info() # Cridem al mètode de la classe pare per mostrar la info bàsica

print(f"Capacitat de la bateria: {self.capacitat\_bateria} kWh")

# Crear objectes i provar l'ampliació

cotxe1 = Cotxe("Toyota", "Corolla", 180)

cotxe1.encendre()

cotxe1.mostra\_info()

cotxe1.apagar()

print("\n--- Cotxe Elèctric ---\n")

cotxe\_el1 = CotxeElectric("Tesla", "Model S", 250, 100)

cotxe\_el1.encendre()

cotxe\_el1.carregar\_bateria()

cotxe\_el1.mostra\_info()

cotxe\_el1.apagar()

**Data actualització i autor/res:** Ignacio López-Aylagas 21/10/2024