Este trabajo tiene licencia CC BY-NC-SA 4.0. Para ver una copia de esta licencia, visite http://creativecommons.org/licenses/by-nc-sa/4.0/

U6P02-Programas_02. Orientación a objetos

1. Descripción

Desarrolla y prueba en Python los programas que permitan resolver los requisitos planteados.

Dentro de la carpeta **PEPXX** crea las subcarpetas **U6/02_orientación_a_objetos .** Para cada programa crea un archivo que contenga: al inicio comentando el enunciado y a continuación la solución.

Puedes crear un repositorio en GitHub con todo el código.

2. Formato de entrega

Será propuesto en clase por el profesor.

3. Trabajo a realizar

Programa01: Herencia básica + uso de super()

Crea un módulo en python que implemete las siguienetes clases:

- Clase `AnimalTerrestre` con:
  - Atributos de instancia: `nombre,edad y peso`
  - Define propiedades (get y set) para los atributos (usando decoradores `@property`)
  - Constructor `__init__(self, nombre, edad, peso)`
  - Método de instancia `saluda` que muestre algo como " `Soy un animal terrestre llamado Kuma y tengo 5 años.`"

- Subclase `Mamifero`
  - Añade el atributo de instancia: `gestacion_dias.`
  - Constructor `__init__(self, nombre, edad, peso, gestacion_dias)`
  - Define propiedades (get y set) para el atributo `(usando decoradores @property)`
  - Sobrescribe saluda para que muestre algo como:"`Soy un mamifero llamado Kuma, tengo 5 años y mi gestación dura 100 días.`"

- Subclase `Ave`
  - Añade el atributo de instancia: `puede_volar (booleano).`
  - Constructor `__init__(self, nombre, edad, peso, puede_volar)`

    - Define propiedades (get y set) para el atributo (usando decoradores `@property`)
    - Sobrescribe `saluda` para que muestre algo como:" `Soy un ave llamado Mau, tengo 2 años y no puedo volar.`

Escribe un programa que importe el módulo y crre una función `main()` que.

- Cree un `Animal`, una `Mamifero` y un `Ave`.
- Llame al método `saluda` para cada uno de ellos.

Programa02: Polimorfismo

Amplia el programa anterior para que la función `main`:

- Cree una lista con varios objetos `Mamifero` y `Ave`.
- Recorra la lista y llame a `saluda` por cada objeto.

Programa03: Clases abstractas y métodos bastractos

Crea un módulo en python que implemete las siguienetes clases:

- Clase abstracta `AnimalMarino` con:
    - Atributos de instancia: `nombre`.
    - Define propiedades (get y set) para el atributo (usando decoradores @property)
    - Método abstracto `sonido()`
    - Método abstracto `saluda()`

- Subclase `Delfin`.
    - Constructor `__init__(self, nombre)`
    - Implementa los métodos abstractos.
        - Método `sonido()`: muestra "`Clicks y silbidos`"
        - Método `saluda()`: muestra algo como "`Soy un delfin llamado Alex.`"

- Subclase `Tiburón`.
    - Constructor `__init__(self, nombre)`
    - Implementa los métodos abstractos
        - Método `sonido()`: muestra "`No tiene un sonido audible característico`"
        - Método `saluda()`: muestra algo como "`Soy un tiburón llamado Mai.`"

Escribe un programa que importe el módulo y crre una función `main` que.

- Cree una lista con varios objetos `Delfin` y `Tiburón`.
- Recorra la lista y llame a los métodos `saluda` y `sonido` por cada objeto.

Programa04: Métodos mágicos "dunder": __str__, comparación y suma

- Amplía la clase `AnimalTerrestre`:
  - Implementa `__str__(self)` en la clase padre y las clases hijas para que `print(animal)` muestre algo como:

```
AnimalTerrestre(nombre='Kuma',edad=5, peso=120.0)
Mamífero(nombre='Kuma',edad=5, peso=120.0)
```

  - Implementa un método mágico de comparación, por ejemplo: `def __lt__(self, otro)` teniendo en cuenta que un animal es 'menor' que otro si su edad es menor.
  - Implementa `__add__(self, otro)` que devuelva **un nuevo animal** que combine a dos:
    - `nombre`: concatenación de nombres ("Kuma-Balto")
    - `edad`: media de ambas edades (entera o float, como prefieras)
    - `peso`: suma de los pesos (si existen, si no, None).

- Crea una función `main` que:
  - Cree varios animales, mamiferos y aves con edad y peso.
  - Los compare con `<` (por ejemplo, `animal1 < animal2`).
  - Cree un "animal combinado" con `animal3 = animal1 + animal2` y lo muestre con `print(animal3)`.

Programa05: Iterables e iteradores

Basándote en los programas anteriores, crea:

- Clase `Manada` con:
  - Atributos de instancia: `lista_animales`.
  - Constructor `__init__(self, lista_animales)`
  - Métodos `añadir(animal)`.
  - Implementa `iter()` y `next()` para iterar la manada animal por animal.

- Crea una función `main` que:
  - Cree una manada.
  - Añada tres animales (de cualquier tipo)
  - Recorra la manada con un `for`.

Programa06: Dependencia, Agregación y Composición

Basándote en los programas anteriores, crea:

- Ejemplo de **dependencia**
  - Una clase utiliza otra solo durante la ejecución de un método.

- No guarda el objeto, no lo conserva, no lo controla.
- Clase `Veterinario` con:
  - Atributos de instancia: `nombre`.
  - `Constructor __init__(self, nombre)`
  - Define propiedades (get y set) para el atributo (usando decoradores `@property`)
  - Método `revisar(animal)` que imprime: "`Revisando a <animal.nombre>2`

- Ejemplo de **agregación**
  - Una clase almacena referencias a objetos creados externamente.
  - No los crea ni los destruye, solo los "tiene"
  - Clase `ReservaNatural` con:
    - Atributos de instancia: `lista_animales`.
    - Constructor `__init__(self, lista_animales)`
    - Métodos `añadir(animal)`.

.

- Ejemplo de **composición**
  - Aquí el todo crea y controla sus partes internas.
  - Las partes no existen fuera del Tiburón.
  - Clase `Corazon` con:
    - Atributos de instancia: `tamaño`.
    - Constructor `__init__(self, tamaño)`
    - Método `late()`: muestra "`estoy latiendo`"
  - Clase `Pulmones` con:
    - Atributos de instancia: `tamaño`.
    - Constructor `__init__(self, tamaño)`
    - Método `respira(): muestra "estoy latiendo"`
  - Clase `Corazon` con:
    - Atributos de instancia: `tamaño`.
    - Constructor `__init__(self, tamaño)`
    - `Método latidos(): muestra "estoy latiendo"`
  - Amplia la clase `Tiburón`.
    - Artributos de instancia:
      - corazón : objeto de la clase `Corazon`.
      - pulmone : objeto de la clase `Pulmones`.
    - Constructor `__init__(self, nombre)`. Crear los objetos de tipo `Corazon` y `Pulmones`.
    - Método `nadar()`. Llama a los métodos late() y respita() del corazon y del pulmon.

- Crea una función `main` que:
  - Cree varios objetos de las clases anteriores y pruebe su comportamiento.