TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Herencia Múltiple
- Sobreescritura de métodos
- Utilizando la función isinstance()

HERENCIA MÚLTIPLE

Es la capacidad de una subclase de heredar de múltiples súper clases.

Esto conlleva un problema, y es que, si varias súper clases tienen los mismos atributos o métodos, la subclase sólo podrá heredar de una de ellas.

En estos casos Python dará prioridad a las clases más a la izquierda en el momento de la declaración de la subclase.

Veamos un ejemplo: por un lado, tenemos dos clases **Clase1** y **Clase2**, y por otro tenemos la **Clase3** que hereda de las dos anteriores. Por lo tanto, heredará todos los métodos y atributos de ambas.

```
1 class Clase1:
2     pass
3 class Clase2:
4     pass
5 class Clase3(Clase1, Clase2):
6     pass
```

Es posible también que una clase herede de otra clase, y a su vez, otra clase herede de la anterior.

```
1 class Clase1:
2     pass
3 class Clase2(Clase1):
4     pass
5 class Clase3(Clase2):
6     pass
```

VENTAJAS

Una de las principales ventajas es la reutilización de código, permitiéndonos establecer una relación entre clases, y evitando que sea necesario volver a declarar ciertos métodos o atributos.

Las clases nos permiten construir objetos sobre una colección de atributos y métodos definidos de forma abstracta. Y la capacidad de herencia nos permitirá crear clases secundarias más grandes, y capaces heredando múltiples atributos y métodos de otras; así como también más específicas controlando los mismos para una única clase particular.

SOBREESCRITURA DE MÉTODOS

Se refiere a la posibilidad de que una subclase cuente con métodos con el mismo nombre que los de una clase superior, pero que definen comportamientos diferentes.

Una subclase sobrescribe un método de su superclase cuando define un método con las mismas características (nombre, número y tipo de argumentos) que el método de la superclase.

La sobreescritura nos permite incorporar un método desde la superclase o clase padre a una clase hija y, aún, modificarlo o alterarlo en ésta última, si es que nos pareciera pertinente. Para ello es requisito indispensable que ambos métodos compartan el mismo nombre. Así, Python llamará, de acuerdo con el flujo de ejecución, al método de la subclase o clase hija en lugar de su homónimo en la superclase o clase padre, cuando dicho flujo de ejecución alcance en su descenso a la clase hija.

UTILIZANDO LA FUNCIÓN ISINSTANCE()

La función `isinstance()` comprueba si el objeto que es el primer argumento es una instancia o subclase de la clase `classinfo` que es el segundo argumento.

- LIBRERÍA:
  - Built-in
- SINTAXIS:
  - isinstance(object, classinfo)
- PARÁMETROS:
  - object: Objeto a evaluar.

- classinfo: nombre de clase, tupla de nombres de clases, o estructura recursiva de tuplas conteniendo nombres de clases.

Podemos comprobar si el número entero 4 pertenece a la clase **int** con el siguiente código:

```
isinstance(4, int)
True
```

Si pasamos como segundo argumento el nombre de la clase str (cadenas de texto), la función devuelve False:

```
isinstance(4, str)
False
```

El argumento **`classinfo`** puede ser una tupla de nombres de clases, devolviendo **True** si el objeto analizado pertenece a alguna de ellas:

```
isinstance(5, (int, str))
True

isinstance(5, (bool, str))
False
```

También puede ser una estructura recursiva de tuplas:

```
isinstance(5, ((int, str), (float, bool)))
True

isinstance(5, ((complex, str), (float, bool)))
False
```

Podemos probar la función con una clase personalizada:

```
class circulo:
    def __init__(self, radio):
        self.radio = radio

c = circulo(3)

isinstance(c, circulo)
True
```

Si el objeto es una instancia de una subclase de la clase indicada, la función también devuelve **True**. Por ejemplo, creamos una subclase de la clase círculo:

```
class circulo_coloreado(circulo):
```

```
    def __init__(self, radio, color):
        super().__init__(radio)
        self.color = color

c_azul = circulo_coloreado(5, "azul")
```

La variable **c_azul** es una instancia de la subclase recién creada, por lo que pasar la variable y el nombre de la subclase a la función **isinstance** devuelve el booleano **True**:

```
isinstance(c_azul, circulo_coloreado)
True
```

Pero pasar a la función el nombre de la variable y la clase padre también devuelve **True**:

```
isinstance(c_azul, circulo)
True
```