EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: CREACIÓN DE UNA HERENCIA MÚLTIPLE
- EXERCISE 2: MÉTODOS ESPECIALES ISINSTANCE() Y __MRO__ PARA CONOCER LA PRESENCIA DE HERENCIA

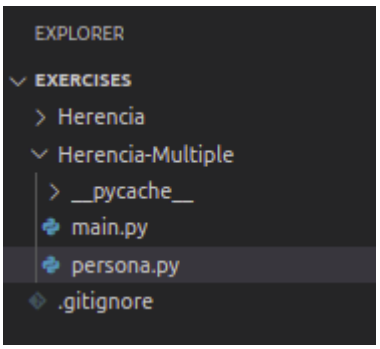
EXERCISE 1: CREACIÓN DE UNA HERENCIA MÚLTIPLE

Continuando con el avance del ejercicio anterior, deseamos crear una nueva clase que contenga y describa las capacidades de un supervisor:

- Atributos: ncertificado, raiting
- Métodos: imprimir_capacidades

Luego, un nuevo objeto de clase **SupervisorZona** contendrá una herencia múltiple entre **Supervisor** y **Capacidades** con un nuevo atributo **promedio**.

Ingresamos a Visual Studio Code (VSC), y procedemos a crear una carpeta llamada Herencia-Múltiple:



Procedemos a crear dos archivos, uno principal llamado `main.py` donde realizaremos el llamado de la clase creada; y otro llamado `persona.py`, donde se definen las clases. Colocaremos la herencia múltiple:

En el archivo `persona.py` procedemos a crear la clase **Capacidades**:

Herencia/persona.py

```
class Capacidades:
    def __init__(self,ncertificados, raiting):
        self.ncertificados = ncertificados
        self.raiting = raiting
```

Seguidamente, creamos el método `imprimir_capacidades` y el método **`__str__`**:

Herencia-Multiple/persona.py

```
class Capacidades:
    def __init__(self,ncertificados, raiting):
        self.ncertificados = ncertificados
        self.raiting = raiting

    def imprimir_capacidades(self):
        print( f'Nro Certificados: {self.ncertificados} \nRaiting:
{self.raiting}')

    def __str__(self):
         return f'Numero de Certificados: {self.ncertificados}
\nRaiting: {self.raiting}'
```

Luego, procedemos a crear la herencia múltiple. La herencia múltiple tiene la capacidad de una subclase heredar de múltiples clases padres o súper clases. El problema radica en que, si las súper clases poseen los mismos atributos o métodos, la subclase creada sólo podrá heredar de una de ellas. Para esto, Python dará prioridad a las clases más a la izquierda en el momento de la declaración de la subclase. Procedemos a crear la subclase múltiple **SupervisorZona**.

Herencia-Multiple/persona.py

```
class SupervisorZona(Supervisor, Capacidades):
    def __init__(self, nombre, apellidos, cedula, zona,
ncertificados, raiting, promedio):
        Supervisor.__init__(self, nombre, apellidos, cedula, zona)
        Capacidades.__init__(self, ncertificados, raiting)
        self.promedio = promedio

    def imprimir_supervisor_zona(self):
        Supervisor.imprimir_supervisor(self)
        Capacidades.imprimir_capacidades(self)
        print('Promedio:',self.promedio)
```

Creamos los objetos de **SupervisorZona** en el archivo `main.py`

Herencia-Multiple/main.py

```
from persona import SupervisorZona

print("******")
supervisorzona1 = SupervisorZona('Juan', 'Pérez', 123456, 'Sur', 8,
25, 123)
supervisorzona1.imprimir_persona()
print("******")
supervisorzona1.imprimir_supervisor()
print("******")
supervisorzona1.imprimir_supervisor_zona()
```

Ejecutando el programa.

Terminal

```
$ python main.py
******
Nombre: Juan
Apellidos: Pérez
Cédula: 123456
******
Nombre: Juan
Apellidos: Pérez
Cédula: 123456
Zona: Sur
******
Nombre: Juan
Apellidos: Pérez
Cédula: 123456
Zona: Sur
Nro Certificados: 8
Raiting: 25
Promedio: 123
```

EXERCISE 2: MÉTODOS ESPECIALES ISINSTANCE() Y __MRO__.

La función `isinstance()` devuelve **True** si el objeto especificado es del tipo especificado; de lo contrario, **False**.

Si el parámetro de tipo es una tupla, esta función devolverá **True** si el objeto es uno de los tipos de la tupla.

Por ejemplo:

Herencia-Multiple/main.py

```
from persona import Persona, Supervisor, SupervisorZona

print("******")
supervisorzona1 = SupervisorZona('Juan', 'Pérez', 123456, 'Sur', 8,
25, 123)
supervisor = Supervisor('Pedro', 'Carrillo', 123456, 'Norte')

print("supervisorzona1")
# Es instancia supervisorzona1 de Persona
print(isinstance(supervisorzona1, Persona))

# Es instancia supervisorzona1 de Supervisor
print(isinstance(supervisorzona1, Supervisor))
```

```
14
15 # Es instancia supervisorzona1 de SupervisorZona
16 print(isinstance(supervisorzona1, SupervisorZona))
17
18 print("supervisor")
19 # Es instancia supervisor de Persona
20 print(isinstance(supervisor1, Persona))
21
22 # Es instancia supervisor de Supervisor
23 print(isinstance(supervisor1, Supervisor))
24
25 # Es instancia supervisor de SupervisorZona
26 print(isinstance(supervisor1, SupervisorZona))
```

Terminal

```
1  $ python main.py
2  ******
3  supervisorzona1
4  True
5  True
6  True
7  supervisor
8  True
9  True
10 False
```

La función `isinstance(objeto, clase)` devuelve **True** si el objeto es de la clase o de una de sus clases hijas. Por tanto, un objeto de la clase **supervisorzona1** es instancia de **SupervisorZona** pero también lo es de **Supervisor** y de **Persona**. Sin embargo, un objeto de la clase **Persona** o **Supervisor** nunca será instancia de la clase **SupervisorZona**.

El siguiente método es el **Method Resolution Order** (**MRO**). Es el orden en el cual el método debe heredar en la presencia de herencia múltiple. Usted puede ver el **MRO** usando el atributo **__mro__**.

Por ejemplo:

Herencia-Multiple/main.py

```
1 from persona import Persona, Supervisor, SupervisorZona
2
3
4 print(SupervisorZona.__mro__)
5 print(Supervisor.__mro__)
6 print(Persona.__mro__)
```

Salida:

Herencia-Multiple/main.py

```
$ python main.py
(<class 'persona.SupervisorZona'>, <class 'persona.Supervisor'>,
<class 'persona.Persona'>, <class 'persona.Capacidades'>, <class
'object'>)
(<class 'persona.Supervisor'>, <class 'persona.Persona'>, <class
'object'>)
(<class 'persona.Persona'>, <class 'object'>)
```