HINTS

HERENCIA MÚLTIPLE

A diferencia de lenguajes como Java y C#, el lenguaje Python permite la herencia múltiple, es decir, se puede heredar de múltiples clases.

La 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:

```
class Destreza(object):
    """Clase la cual representa la Destreza de la Persona"""

    def __init__(self, area, herramienta, experiencia):
        """Constructor de clase Destreza"""
        self.area = area
        self.herramienta = herramienta
        self.experiencia = experiencia

    def __str__(self):
        """Devuelve una cadena representativa de la Destreza"""
        return """Destreza en el área %s con la herramienta %s,
        tiene %s años de experiencia.""" % (
            str(self.area), self.experiencia, self.herramienta)


class JefeCuadrilla(Supervisor, Destreza):
    """Clase la cual representa al Jefe de Cuadrilla"""

    def __init__(self, cedula, nombre, apellido, sexo,
        rol, area, herramienta, experiencia, cuadrilla):
        """Constructor de clase Jefe de Cuadrilla"""

        # Invoca al constructor de clase Supervisor
        Supervisor.__init__(self, cedula, nombre, apellido, sexo,
            rol)
        # Invoca al constructor de clase Destreza
        Destreza.__init__(self, area, herramienta, experiencia)

        # Nuevos atributos
        self.cuadrilla = cuadrilla

    def __str__(self):
        """Devuelve cadena representativa al Jefe de Cuadrilla"""
```

```
        jq = "{0}: {1} {2}, rol '{3}', tareas {4}, cuadrilla: {5}"
        return jq.format(
            self.__doc__[28:46], self.nombre, self.apellido,
            self.rol, self.consulta_tareas(), self.cuadrilla)
```

METHOD RESOLUTION ORDER (MRO)

Ese 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__**.

```
>>> JefeCuadrilla.__mro__
(<class '__main__.JefeCuadrilla'>,
<class '__main__.Supervisor'>,
<class '__main__.Persona'>,
<class '__main__.Destreza'>,
<type 'object'>)
>>> Supervisor.__mro__
(<class '__main__.Supervisor'>,
<class '__main__.Persona'>,
<type 'object'>)
>>> Destreza.__mro__
(<class '__main__.Destreza'>,
<type 'object'>)
```

El **MRO** es calculado en Python de la siguiente forma:

Un método en la llamada derivada es siempre llamado antes del método de la clase base. En nuestro ejemplo, la clase `JefeCuadrilla` es llamada antes de las clases `Supervisor` o `Destreza`. Esas dos clases son llamadas antes de la clase `Persona`, y la clase `Persona` es llamada antes de la clase `Object`.

Si hay herencia múltiple como `JefeCuadrilla(Supervisor, Destreza)`, el método invoca a `Supervisor` primero porque ese aparece primero de izquierda a derecha.