Práctica Herencia

La forma en la que programé la herencia es un poco diferente a la que se mostró en clase, el método que empleé hace uso del MRO (Method Resolution Order) que determina orden en que los atributos y métodos se añaden en una nueva clase con herencia múltiple, también evita repetir la ejecución de clases padre. Con este método ya no es necesario especificar todos los atributos heredados como argumentos del __init__ de una clase hijo, solo es necesario pasar los argumentos *args y **kwargs a la función super.

Para que este método funcione agregué una clase llamada obj que solamente termina el proceso. En la explicación de cada clase agregaré una breve explicación de qué es lo que hace el MRO para llevar a cabo la herencia.

En todos los atributos de todas las clases está por defecto el valor "DESCONOCIDO", esto en caso de que al crear un objeto no se pasen como argumentos todos los atributos.

Clase persona

```python
class persona(obj):
    def __init__(self,
                 nombre = "DESCONOCIDO",
                 sexo = "DESCONOCIDO",
                 edad = "DESCONOCIDO",
                 lugNac = "DESCONOCIDO",
                 *args,
                 **kwargs):
        self.nombre = nombre
        self.sexo = sexo
        self.edad = edad
        self.lugNac = lugNac
        super(persona,self).__init__(*args, **kwargs)

    def info(self):
        print(
            "\nNombre: ", self.nombre,
            "\nSexo: ", self.sexo,
            "\nEdad: ", self.edad,
            "\nLugar de nacimiento: ", self.lugNac
        )
```

Como parámetros de entrada tenemos nombre, sexo, edad y lugar de nacimiento. El único método programado es info() que imprime los atributos en forma de lista.

La función super en esta clase solo es útil cuando herede a otra clase, pero no afecta a su funcionamiento si no hereda, en la explicación de la clase alumno se explica su funcionamiento.

Clase Destreza

```
class destreza(obj):
    def __init__(self,
                 mejor_materia = "DESCONOCIDA",
                 peor_materia = "DESCONOCIDA",
                 *args,
                 **kwargs):
        self.mejor_materia = mejor_materia
        self.peor_materia = peor_materia
        super(destreza, self).__init__(*args, **kwargs)

    def info(self):
        print(
            "\nMejor materia: ", self.mejor_materia,
            "\nPeor materia: ", self.peor_materia
        )
```

Como parámetros de entrada tenemos mejor_materia y peor_materia. El único método programado es info() que imprime los atributos en forma de lista.

El funcionamiento de super es igual que en la clase persona. En la siguiente clase se explica el funcionamiento de la herencia.

Clase alumno

```
class alumno(persona, destreza):
    def __init__(self,
                 NUA = "DESCONOCIDO",
                 semestre = "DESCONOCIDO",
                 promedio = "DESCONOCIDO",
                 carrera = "DESCONOCIDO",
                 *args,
                 **kwargs):
        self.NUA = NUA
        self.semestre = semestre
        self.promedio = promedio
        self.carrera = carrera
        super(alumno,self).__init__(*args, **kwargs)

    def info(self):
        print("\n")
        persona.info(self)
        destreza.info(self)
        print(
            "\nNUA: ", self.NUA,
            "\nSemestre: ", self.semestre,
            "\nPromedio: ", self.promedio,
            "\nCarrera: ", self.carrera
        )
```

Como parámetros de entrada tenemos NUA, semestre, promedio y carrera.

Los argumentos *args y **kwargs sirven para recibir los valores de los atributos de las clases de las que hereda, que son persona y destreza. Estos argumentos se pasan a la función super que lo que hará es ejecutar la función __init__ de la primera clase de la que hereda, que es persona, luego, dentro del __init__ de la clase persona la función super que se encuentra ahí ejecutará la función __init__ de la siguiente clase que hereda, que es destreza.

Al final de este proceso se habrán agregado los atributos y métodos de las dos clases padre, y en caso de repetirse se tomará aquel atributo o método de la última clase en la que se ejecutó el __init__. Después de eso se crean los métodos de la clase alumno, sobreponiéndose en caso de repetirse con alguno ya agregado por herencia.

El único método programado es info() que imprime todos los atributos de la clase, primero los atributos heredados mediante los métodos de las clases padre y luego los atributos propios.

Clase profesor

```
class profesor(persona, destreza):
    def __init__(self,
                 NUE = "DESCONOCIDO",
                 antiguedad = "DESCONOCIDO",
                 horasLab = "DESCONOCIDO",
                 grado = "DESCONOCIDO",
                 *args,
                 **kwargs):
        self.NUE = NUE
        self.antiguedad = antiguedad
        self.horasLab = horasLab
        self.grado = grado
        super(profesor, self).__init__(*args, **kwargs)

    def info(self):
        print("\n")
        persona.info(self)
        destreza.info(self)
        print(
            "\nNUE: ", self.NUE,
            "\nAntigüedad: ", self.antiguedad,
            "\nHoras laboradas: ", self.horasLab,
            "\nGrado: ", self.grado
        )
```

Como parámetros de entrada tenemos NUE, antigüedad, horas laboradas y grado.El funcionamiento de la herencia y el método info() son lo mismo que la clase alumno.

Tal y como están escritas las herencias, no es necesario que la clase destreza tenga la función super incluida, al ser la última clase de la que será ejecutada el __init__, la función super ya no tiene la necesidad de redireccionar la ejecución a otro __init__ de otra clase. Pero en caso de que en el futuro programemos alguna otra clase en la que destreza no sea la última en heredar, funcionará.

Clase obj

```
class obj():
    def __init__(self):
        super(obj,self).__init__()
```

Esta clase es padre de aquellas que en nuestro esquema no tienen padre, solamente sirve para ejecutar la función super sin argumentos, lo que permite terminar el proceso sin problemas y dejar listas las clases para heredar en cualquier orden sin necesidad de cambiar sus argumentos en la función super.

Ejemplos

```
alumno1 = alumno(427086,
                 8,
                 9.01,
                 "Licenciatura en Física",
                 "Uriel Chávez Flores",
                 "Masculino",
                 21,
                 "Aguascalientes, Aguascalientes",
                 "POOyE",
                 "Cosmología")
alumno1.info()
```

```
profesor1 = profesor(301580,
                     4,
                     15,
                     "Maestría",
                     "Eric Ruiz Flores",
                     "Masculino",
                     24,
                     "Aguascalientes, Aguascalientes",
                     "Sistemas Lineales",
                     "POOyE")
profesor1.info()
```

```
alumno2 = alumno(sexo = "Femenino",
                 NUA = 426062,
                 peor_materia = "Métodos Numéricos",
                 semestre = 10,
                 lugNac = "Leon, Gto.",
                 nombre = "Fernanda Guerrero Aguilar",
                 mejor_materia = "Análisis Tensorial")
alumno2.info()
```

Podemos ingresar los argumentos sin orden específico igualando el nombre del argumento a algún valor como se muestra en el tercer ejemplo. Si ingresamos menos argumentos de los necesarios se tomará como valor “DESCONOCIDO” que es el valor por defecto que especificamos en cada clase.

Ejecución

```
Nombre:  Uriel Chávez Flores
Sexo:  Masculino
Edad:  21
Lugar de nacimiento:  Aguascalientes, Aguascalientes

Mejor materia:  POOyE
Peor materia:  Cosmología

NUA:  427086
Semestre:  8
Promedio:  9.01
Carrera:  Licenciatura en Física



Nombre:  Eric Ruiz Flores
Sexo:  Masculino
Edad:  24
Lugar de nacimiento:  Aguascalientes, Aguascalientes

Mejor materia:  Sistemas Lineales
Peor materia:  POOyE

NUE:  301580
Antigüedad:  4
Horas laboradas:  15
Grado:  Maestría



Nombre:  Fernanda Guerrero Aguilar
Sexo:  Femenino
Edad:  DESCONOCIDO
Lugar de nacimiento:  Leon, Gto.

Mejor materia:  Análisis Tensorial
Peor materia:  Métodos Numéricos

NUA:  426062
Semestre:  10
Promedio:  DESCONOCIDO
Carrera:  DESCONOCIDO
```