UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

HERENCIA MÚLTIPLE EN PYTHON

Javier Alejandro Sanchez Salamanca - 20201020167

Jefferson Escobar Rivas - 20192020143

Jonathan Sneyder Ardila Neira - 20191007058

Docente: SIMAR ENRIQUE HERRERA JIMÉNEZ

MODELOS DE PROGRAMACIÓN 2

GRUPO 81

FACULTAD DE INGENIERÍA

BOGOTÁ D.C

2022

INTRODUCCIÓN;

En la actualidad se escucha hablar de técnicas, procedimientos o guías adecuadas para programar un buen código, dependiendo del objetivo de este, lo que conlleva al surgimiento de los "Modelos de Programación", donde a partir de estos, podemos evaluar la "Herencia Simple" o "Herencia Múltiple", que permiten tener una clase padre y poder derivar subclases hijas que dependan métodos y atributos de esta clase padre.

MARCO TEÓRICO

La herencia de clases es una técnica de la programación orientada a objetos (POO) muy útil que permite crear una clase general (Clase base) primero y luego crear "subclases" (Clases derivadas) más especificas que re-utilicen el código de la clase general.

La sintaxis para la definición de una clase derivada es la siguiente:

```
1.  class ClaseDerivada(ClaseBase):
2.    '''
3.
4.    Contenido de la clase ClaseDerivada
5.
6.    '''
```

ClaseBase debe ser accesible desde el lugar donde se está definiendo **ClaseDerivada**. En caso de estar en un módulo distinto, se indica cuál es la clase base de la forma modulo.**ClaseBase.**

Herencia Múltiple

La herencia múltiple ocurre cuando una clase es derivada de dos clases base o más. Las clases base se indican de la misma forma, separando cada una con una coma.

```
class ClaseDerivada(ClaseBase1,ClaseBase2):
  '''
  Contenido de la clase
  '''
```

Las clases derivadas heredan todos los atributos y métodos de las clases que tomen como base. Continuando con el ejemplo de los empleados, creemos una clase mas que será incluida en el esquema.

```
class Empleado(object):
  # Constructor de la clase
  def __init__(self,nombre,iden):
    self.nombre = nombre
    self.iden = iden

class Valencia(object):
  domicilio = "Valencia"

class RecHumanos(Empleado,Valencia):
  def saludo(self):
    print("Hola, mi nombre es " + self.nombre + " y mi ID es " + self.iden + ".")
    print("Trabajo en Recursos Humanos.")
    print("Vivo en " + self.domicilio + ".")
```

La clase consiste en el lugar de residencia del empleado. En este caso, la clase es para los empleados cuya residencia sea Valencia. Al crear el objeto de nuevo y utilizar la función de saludo se obtendrá lo siguiente:

```
>>> Karla = RecHumanos("Karla","182052")
>>> Karla.saludo()
Hola, mi nombre es Karla y mi ID es 182052.
Trabajo en Recursos Humanos.
Vivo en Valencia.
```

La práctica de la herencia de clases es utilizada principalmente para poder reutilizar código estableciendo una relación entre clases, lo que evita la necesidad de declarar mas de una vez algún método o atributo de alguna clase.

El uso de clases nos permite crear objetos con determinados atributos y métodos definidos con cierta abstracción. La herencia de clases nos permitirá crear clases secundarias, mas específicas, que obtengan atributos y métodos de otras.

EJEMPLO

```
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)
```

La herencia múltiple es la capacidad de una subclase de heredar de múltiples superclases. Esto conlleva un problema, y es que si varias superclases tienen los mismos atributos o métodos, la subclase solo 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:

CONCLUSIONES

Después de la investigación realizada, podemos deducir que a diferencia de lenguajes como Java y C#, el lenguaje Python permite la herencia múltiple, es decir, se puede heredar de múltiples clases, ya que permite tener clases derivadas que heredan los atributos y métodos de las clases que tomen como base, para así hacer más funcional el código y poder tener procedimientos más efectivos.

WEBGRAFÍA

- https://unipython.com/herencia-multiple-de-clases-en-python/#:~:text=La%20herencia%20m%C3%BAltiple%20ocurre%20cuando,cada%20una%20con%20una%20coma.&text=Las%20clases%20derivadas%20heredan%20todos,clases%20que%20tomen%20como%20base.

- https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion9/herencia.html