Programación Orientada a Objetos

Herencia

Índice

- Concepto
- Constructores en la herencia
- Funciones para trabajar con herencia
- Herencia múltiple

Concepto

- La herencia permite crear clases a partir de otras existentes, heredando los atributos y métodos de la clase padre o superclase
- Las clases de Python pueden heredar más de una clase (herencia múltiple)
- En la definición de la clase, se indica entre paréntesis el nombre de la clase o clases que se heredan:

```
class ClaseHija(ClasePadre):
    …

class ClaseHija(ClasePadre1, ClasePadre2):
    …
```

Herencia múltiple

Constructores en la herencia

- Las clases hijas o subclases, heredan los constructores de las clases padre o superclases, aunque pueden definir el suyo propio:

```
class Padre:
    def __init__(self):
        print("constructor padre")
class Hija(Padre):
    pass
h1=Hija()
```

Imprime *constructor padre*

```
class Padre:
    def __init__(self):
        print("constructor padre")
class Hija(Padre):
    def __init__(self):
        print("constructor hija")
h1=Hija()
```

Imprime *constructor hija*

- Se puede llamar desde el constructor de la subclase al de la superclase con:

```
Padre.__init__(self)
```

Funciones para trabajar con herencia

- Python proporciona dos funciones para comprobar la relación de herencia entre clases e instancias

issubclass(sub, sup).	• Indica si la primera clase es o no sublcase de la segunda
isinstance(obj, clase).	• Indica si la referencia a la instancia que se pasa como primer parámetro es un objeto de la clase indicada en segundo parámetro o de alguna de sus subclases

- Ejemplo:

```
class Padre:
    pass
class Hija(Padre):
    pass
h1=Hija()
print(issubclass(Hija,Padre))  #True
print(isinstance(h1,Padre))   #True
```

Herencia múltiple

- La herencia consiste en heredar más de una clase
- Si se heredan dos versiones de un mismo método desde dos clases, prevalece la de la primera clase de la lista:

```
class Clase1:
    def m1():
        print("m1 de Clase1")
```

```
class Clase2:
    def m1():
        print("m1 de Clase2")
```

```
class ClaseNueva(Clase1, Clase2):
    pass

obj=ClaseNueva()
obj.m1() #imprime m1 de Clase1
```