**HERENCIA**

La herencia se refiere a crear clases “hijas” desde clases “padres”.

El ejemplo que se propone es el de la clase Empleado y la clase Persona, donde la segunda es la clase padre de la primera.

Para indicar herencia en Python se utiliza la siguiente sintaxis:

**Class(Padre) Hija:**

Además, cuando se “inicia” una instancia de la clase hija, también se deben “cumplir los requisitos” de la clase padre, esto significa que se deben inicializar los atributos dispuestos en el método \_\_init\_\_ de la clase padre.

class Empleado(Persona):

    def \_\_init\_\_(self, nombre, apellidos, edad, sueldo, empresa):

        super().\_\_init\_\_(nombre, apellidos, edad)

        self.\_nombre = nombre

        self.\_apellidos = apellidos

        self.\_edad = edad

        self.\_sueldo = sueldo

        self.\_empresa = empresa

Este es un código de ejemplo de herencia de clases en Python. Como se ve, se define el \_\_init\_\_ de la clase hija con los atributos de la clase padre (nombre, apellidos, edad) y los atributos **propios de la clase hija, de los cuales no dispone la clase padre (sueldo y empresa)**

En la línea siguiente se utiliza super() para hacer referencia a la clase padre y se accede al método \_\_init\_\_ de la misma, en el cual se “requieren” los atributos de la clase padre.

Por ultimo, este sería un ejemplo valido de la creación de una instancia a partir de la clase hija.

peon = Empleado('Juan', 'Perez', 29, 80000, 'Globant')

**SOBREESCRIBIENDO EL METODO \_\_STR\_\_**

Este método es el que se encarga de imprimir la dirección de memoria de un objeto al momento de realizar un: **print(object)**

En este caso, sobreescribir el método sobreescribe el comportamiento del mismo. Y si sobreescribiesemos el método en la clase padre, también los cambios se verían reflejados en la clase hija.

Ejemplo:

En la clase padre se sobreescribe el método \_\_str\_\_() de la siguiente manera:

    def \_\_str\_\_(self) -> str:

        return f'Persona-> [ Nombre: {self.\_nombre}, Apellidos: {self.\_apellidos}, Edad: {self.\_edad}]'

Esto hace que cada vez que vayamos a realizar un print() de un elemento de tipo Persona obtengamos el siguiente resultado:



Ahora, si haríamos lo mismo con los objetos de tipo Empleado, no recibiríamos los atributos propios de esta clase. (sueldo y empresa), para eso es necesario reescribir el método en la clase hija de esta forma:

    def \_\_str\_\_(self) -> str:

        return f'Empleado-> [Sueldo: {self.\_sueldo}, Empresa: {self.\_empresa}] {super().\_\_str\_\_()} ]'

En este caso se imprime la Cadena Empleado con los atributos Sueldo y Empresa, pero para acceder a los mismos valores que accede la clase padre se vuelve a utilizar **super()** (para hacer referencia a la clase padre) y se accede al método \_\_str\_\_() que se encarga de recuperar los mismos atributos e imprimirlos, por tanto el resultado en este caso sería:

