#clase

class Auto:

#Atributos generales

marca = ""

modelo = 0

placa = ""

#Elementos que puedo añadir a la clase

#Vamos a Añadir un objeto llamado Taxi

taxi = Auto()

#Para llamarlo colocamos el nombre del objeto

print(taxi.modelo)

# Antes de la version python3 class Persona(0bject):

class Persona:

#objetos

doctor = 'Victor'

print(Persona.doctor)

# Antes de la version python3 class Persona(0bject):

class Jugadores\_A:

#objetos

j1 = 'Messi'

j2 = 'C.Ronaldo'

class Jugadores\_B:

#objetos

j1 = 'Marcelo'

j2 = 'Falcao'

print(Jugadores\_A.j1, Jugadores\_A.j2)

print(Jugadores\_B.j1, Jugadores\_B.j2)

# Antes de la version python3 class Persona(0bject):

class Nombre:

pass

# Creacion de objetos

victor = Nombre()

maria = Nombre()

# Se llama el objeto y se crea el atributo

# objeto.atributo = valor

victor.edad = 30

victor.sexo = 'Masculino'

victor.pais = 'Bolivia'

maria.edad = 25

maria.sexo = 'Femenino'

maria.pais = 'Colombia'

print(victor.edad, victor.sexo, victor.pais)

print(maria.edad, maria.sexo, maria.pais)

class Matematica:

#Creación de métodos

#self hace referencia a un determinado objeto

def suma(self, n1, n2):

self.n1 = 0

self.n2 = 0

n3 = n1 + n2

return n3

# Creación del objeto para acceder al método

s = Matematica()

a = int(input("Ingrese un valor entero: "))

b = int(input("Ingrese otro valor entero: "))

# Llamado del método

sum = s.suma(a,b)

# Impresión del resultado

print(sum)

class Ropa:

#Creacion de métodos

#self hace referencia a un determinado objeto

# init significa inicializar

def \_\_init\_\_(self):

# Asignar atributos

self.marca = input("Ingrese la marca: ")

self.talla = input("Ingrese la talla: ")

self.color = input("Ingrese la el color: ")

# Creacion del objeto para acceder al método

camisa = Ropa()

# Impresión del resultado

print("\nHola, de la camisa, ha seleccionado: \nMarca:",camisa.marca,"\nTalla:",camisa.talla,"\nColor:",camisa.color)

class Calculadora:

#Creacion de métodos

#self hace referencia a un determinado objeto

# init significa inicializar

def \_\_init\_\_(self, n1, n2):

# Asignar atributos

self.suma = (n1 + n2)

self.resta = (n1 - n2)

self.mult = (n1 \* n2)

self.div = (n1 / n2)

ope = int(input("INGRESE LA POPERACION DESEADA:\nPARA LA SUMA 1\nPARA LA RESTA 2\nPARA LA MULTIPLICACION 3\nPARA LA DIVISION 4\n"))

# Impresión del resultado

while ope > 0 and ope <= 4:

num1 = float(input("Ingrese el primer valor: "))

num2 = float(input("Ingrese el segundo valor: "))

# Creacion del objeto para acceder al método

operacion = Calculadora(num1, num2)

if ope == 1:

print("La Suma de",num1,"+",num2,"es:",operacion.suma,"\n")

elif ope == 2:

print("La Resta de",num1,"-",num2,"es:",operacion.resta,"\n")

elif ope == 3:

print("La Multiplicacion de",num1,"\*",num2,"es:",operacion.mult,"\n")

elif ope == 4:

print("La División de",num1,"/",num2,"es:",operacion.div,"\n")

opcion = input("Desea realizar otra operacion? S/N:")

opcion = opcion.upper()

if(opcion == 'S'):

ope = int(input("INGRESE LA POPERACION DESEADA:\nPARA LA SUMA 1\nPARA LA RESTA 2\nPARA LA MULTIPLICACION 3\nPARA LA DIVISION 4\n"))

else:

ope = int(input("Para salir de la aplicación presione un numero diferente a las opciones de las operaciones:"))

print("Usted ha salido de la aplicación, adios...")

# Funciones para atributos

class Persona():

edad = int(input("Ingrese la edad: "))

nombre = input("Ingrese el nombre: ")

pais = input("Ingrese el pais: ")

# Crear un objeto

doctor = Persona()

# Llamar a los atributos mediante el objeto creado

print("\nHola",doctor.nombre,"Tienes",doctor.edad,"años, de nacionalidad",doctor.pais)

# Uuso de funciones

print("\nHola",getattr(doctor, 'nombre'),"Tienes",getattr(doctor, 'edad'),"años.","de nacionalidad",getattr(doctor, 'pais'))

# funcion que devuelve si existe o no algún atributo

print("La variable edad existe?", hasattr(doctor, 'edad'))

print("La variable nombre existe?", hasattr(doctor, 'nombre'))

print("La variable pais existe?", hasattr(doctor, 'pais'))

#Funcion que cambia informacion de los atributos

setattr(doctor, 'nombre', 'Andrea')

print("El nuevo nombre es:",doctor.nombre)

# Para eliminar un atributo dentro de una clase

delattr(Persona, 'pais')

print("La variable pais existe?", hasattr(doctor, 'pais'))

# Método contructor

class Persona():

# Como la clase no tiene atributos, colocamos pass

pass

# Primer método, constructor \_\_init\_\_

def \_\_init\_\_(self, nombre, año):

# Atributos del método

self.nombre = nombre

self.año = año

# Creacion de métodos

def descripcion(self):

# format, es usado para mostrar el contenido de los atributos, en el orden ue se ingresan en el parentesis

return '{}, tiene {} años'.format(self.nombre, self.año)

def comentario(self, frase):

# Funciones para atributos

return '{} dice: {}'.format(self.nombre, frase)

n = input('Ingrese su Nombre: ')

a = int(input('Ingrese su edad: '))

# Creacion del objeto

doctor = Persona(n,a)

print(doctor.descripcion())

print(doctor.comentario('Hola que tal!'))

# Método constructor

# Modificar un atributo

class Email:

# Definir el constructor

def \_\_init\_\_(self):

self.enviado = False

def enviar\_correo(self):

self.enviado = True

# Creacion del Objeto

mi\_correo = Email()

# Mostrar atributos del constructor

print(mi\_correo.enviado)

# mostrar variable de un método cualquiera

mi\_correo.enviar\_correo()

print(mi\_correo.enviado)

HERENCIA:

# Herencia

# Clase Padre

class Pokemon:

pass

# Creacion del constructor

def \_\_init\_\_(self, nombre, tipo):

self.nombre = nombre

self.tipo = tipo

def descripcion(self):

return '{}, es un pokemon de tipo: {}'.format(self.nombre, self.tipo)

# Clase hija

class Pikachu(Pokemon):

def ataque(self, tipo\_ataque):

return 'El pokemon Pikachu llamado {}, tiene el tipo de ataque: {}'.format(self.nombre, tipo\_ataque)

# Clase hija

class Charmander(Pokemon):

def ataque(self, tipo\_ataque):

return 'El pokemon Charmander llamado {}, tiene el tipo de ataque: {}'.format(self.nombre, tipo\_ataque)

n = input('Ingrese el nombre del pokemon: ')

t = input('Ingrese el tipo del pokemon: ')

a = input('Ingrese el tipo de ataque del pokemon: ')

# creacion del objeto, haciendo llamado a la clae Pikachu

nuevo\_pokemon = Pikachu(n, t)

# Visualizar el objeto nuevo\_pokemon

print(nuevo\_pokemon.descripcion())

# Visualizar ataque de la clase pikachu

print(nuevo\_pokemon.ataque(a))

# creacion del objeto

nuevo\_pokemon\_1 = Charmander(n, t)

# Visualizar el objeto

print(nuevo\_pokemon\_1.descripcion())

# Visualizar ataque de la clase pikachu

print(nuevo\_pokemon\_1.ataque(a))

# EJEMPLO PRACTICO DE HERENCIA

# Clase Padre

class Calculadora:

# Introducimos el constructor

def \_\_init\_\_(self, numero):

# Caracteristicas o atributos de la calculadora

self.n = numero

# se crea un bucle para iterar desde cero hasta el número que hayamos escogido, como cantidad de operandos

self.datos = [0 for i in range(numero)]

# Método para llamar los datos

def ingresardato(self):

# ingresar los valores a operar

self.datos = [int(input('Ingrese dato '+ str(i+1)+ ' = ')) for i in range(self.n)]

#Clase Hija

class Ope\_Basicas(Calculadora):

def \_\_init\_\_(self):

# Se llama a la clase padre y se limita la entrada de datos a dos

Calculadora.\_\_init\_\_(self,2)

def suma(self):

a,b, = self.datos

s = a + b

print('Suma {} + {} = {}'.format(a, b, s))

def resta(self):

a,b, = self.datos

r = a - b

print('Resta {} - {} = {}'.format(a, b, r))

def mult(self):

a,b, = self.datos

m = a \* b

print('Multiplicación {} \* {} = {}'.format(a, b, m))

def div(self):

a,b, = self.datos

d = float(a / b)

print('Resta {} / {} = {}'.format(a, b, round(d,2)))

# Clase hija

class Raiz(Calculadora):

def \_\_init\_\_(self):

# Se llama a la clase padre y se limita la entrada de datos a uno

Calculadora.\_\_init\_\_(self,1)

def cuadrada(self):

import math # Permite usar funciones directas a travez de librerias

a, = self.datos

print('Raiz {} ^ (1/2) = {}: '.format(a, round(math.sqrt(a),2)))

# Ejecutar el código

calculo = int(input('Ingrese el cálculo que desea realizar:\n1. Operaciones Básicas\n2. Raíz\n\n'))

if calculo == 1:

op\_bas = Ope\_Basicas()

print(op\_bas.ingresardato())

op = int(input('Ingrese la operación a realizar:\n1. Suma\n2. Resta\n3. Multiplicación\n4. Division\n\n'))

if op == 1:

print(op\_bas.suma())

elif op == 2:

print(op\_bas.resta())

elif op == 3:

print(op\_bas.mult())

elif op == 4:

print(op\_bas.div())

elif calculo == 2:

raiz\_cuadrada = Raiz()

print(raiz\_cuadrada.ingresardato())

print(raiz\_cuadrada.cuadrada())

# EJEMPLO PRACTICO DE HERENCIA - FUNCION INTEGRADA (PRUEBAS)

# Clase Padre

class Calculadora:

# Introducimos el constructor

def \_\_init\_\_(self, numero):

# Caracteristicas o atributos de la calculadora

self.n = numero

# se crea un bucle para iterar desde cero hasta el numero que hayamos escogido, como cantidad de operandos

self.datos = [0 for i in range(numero)]

# Método para llamar los datos

def ingresardato(self):

# ingresar los valores a operar

self.datos = [int(input('Ingrese dato '+ str(i+1)+ ' = ')) for i in range(self.n)]

#Clase Hija

class Ope\_Basicas(Calculadora):

def \_\_init\_\_(self):

# Se llama a la clase padre y se limita la entrada de datos a dos

Calculadora.\_\_init\_\_(self,2)

def suma(self):

a,b, = self.datos

s = a + b

print('Suma {} + {} = {}'.format(a, b, s))

def resta(self):

a,b, = self.datos

r = a - b

print('Resta {} - {} = {}'.format(a, b, r))

def mult(self):

a,b, = self.datos

m = a \* b

print('Multiplicación {} \* {} = {}'.format(a, b, m))

def div(self):

a,b, = self.datos

d = float(a / b)

print('Resta {} / {} = {}'.format(a, b, round(d,2)))

# Clase hija

class Raiz(Calculadora):

def \_\_init\_\_(self):

# Se llama a la clase padre y se limita la entrada de datos a uno

Calculadora.\_\_init\_\_(self,1)

def cuadrada(self):

import math # Permite usar funciones directas a travez de librerias

a, = self.datos

print('Raiz {} ^ (1/2) = {}: '.format(a, round(math.sqrt(a),2)))

# Ejecutar el código

calculo = int(input('Ingrese el cálculo que desea realizar:\n1. Operaciones Básicas\n2. Raíz\n'))

if calculo == 1:

op\_bas = Ope\_Basicas()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Verifica la Herencia

test = isinstance(op\_bas, Ope\_Basicas)

print('Se está aplicando Herencia en la clase Ope\_Basicas: ',test)

# Verifica Herencia de clase

test1 = issubclass(Ope\_Basicas, Calculadora)

print('Se está Heredando de la clase Calculadora: ',test1,'\n')

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print(op\_bas.ingresardato())

op = int(input('Ingrese la operación a realizar:\n1. Suma\n2. Resta\n3. Multiplicación\n4. Division\n'))

if op == 1:

print(op\_bas.suma())

elif op == 2:

print(op\_bas.resta())

elif op == 3:

print(op\_bas.mult())

elif op == 4:

print(op\_bas.div())

elif calculo == 2:

raiz\_cuadrada = Raiz()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Verifica la Herencia

test = isinstance(raiz\_cuadrada, Raiz)

print('Se está aplicando Herencia en la clase Raiz: ',test)

# Verifica Herencia de clase

test1 = issubclass(Raiz, Calculadora)

print('Se está Heredando de la clase Calculadora: ',test1,'\n')

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print(raiz\_cuadrada.ingresardato())

print(raiz\_cuadrada.cuadrada())

# HERENCIA MULTIPLE

# CONSISTE EN CREAR UNA CLASE A PARTIR DE MULTIPLES CLASES SUPERIORES

# HERENCIA MULTINIVEL

# CONSISTE EN QUE LA CLASE BASE Y LA DERIVADA, SE HEREDAN EN LA NUEVA DERIVADA

class Telefono:

# Constructor de la clase Telefono

def \_\_init\_\_(self):

pass

def llamar(self):

print('Llamando...')

def ocupado(self):

print('Ocupado...')

class Camara:

# Constructor de la clase Camara

def \_\_init\_\_(self):

pass

def fotografia(self):

print('Tomando fotos...')

class Reproduccion:

# Constructor de la clase Reproduccion

def \_\_init\_\_(self):

pass

def reproduccion\_Musica(self):

print('Reproduciendo música...')

def reproducir\_Video(self):

print('Reproduciendo video...')

# Herencia ultiple: Consiste en tener una clase, la cual enlace, las clases superiores

class Smart\_Phone(Telefono, Camara, Reproduccion):

def \_\_del\_\_(self):

print('Telefono apagado...')

# Eliminar de memoria cuando no estan en uso (destructor)

# Se usa como ahorro de memoria para optimizar ejercicios

# Se crea el objeto para hacer el llamado a las clases

# Se llama a la clase Smart Phone, porque en esta clase ya se encuentran las 3 clases

movil = Smart\_Phone()

# Para saber qué métodos especiales, podemos utilizar o qué acciones puedo aplicarle al objeto

# print(dir(movil))

print(movil.llamar())

print(movil.ocupado())

print(movil.fotografia())

print(movil.reproduccion\_Musica())

print(movil.reproducir\_Video())

print(movil.\_\_del\_\_())

# str.format()

# .format %

# f-strings

curso = 'python'

print('tutoriales de % s'%curso)

# Ejemplo

nombre = 'Victor'

edad = 25

# str.format()

print("Hola, soy % s, y tengo % s años."%(nombre, edad))

# .format %

print("Hola que tal, soy {} y tengo {} años.".format(nombre, edad))

# f-strings

print(f"hola, soy {nombre}, y mi edad es {edad} años")

# f-strings

class Estudiante:

def \_\_init\_\_(self, nombre, apellido, edad):

self.nombre = nombre

self.apellido = apellido

self.edad = edad

'''

# Representacion informal de una cadena o de un objeto

def \_\_str\_\_(self):

# Usamos str para poder visualizar y ejecutar un resultado inmediatamente

return f"Hola que tal, soy {self.nombre} {self.apellido}, tengo {self.edad} años."

'''

# Representacion oficial no ambigua y que se ejecuta con el código

def \_\_repr\_\_(self):

# Tiene en cuenta el valor de las variables

return f"Hola que tal, soy {self.nombre} {self.apellido}, tengo {self.edad} años."

# Cuando creamos el objeto, este va a contener los valores como tal y no una cadena de sting

n = input("Ingrese su nombre: ")

a = input("Ingrese su apellido: ")

e = input("Ingrese su edad: ")

# Se crea el objeto

nuevo\_estudiante = Estudiante(n, a, e)

# Se hace llamado a la funcion String

# print(f"{nuevo\_estudiante}")

# Se hace llamado a la funcion String y al método rep

print(f"{nuevo\_estudiante !r}")

Diferencias entre una Clase y una Instancia:

**Una clase**: es una plantilla para crear objetos, o un molde para crear distintos objetos, similar a los planos para construir una casa.

**Una Instancia:**

Un Objeto se crea usando el constructor de una clase.

Una vez que el objeto es creado, se le conoce como una instancia de la clase.

En Python, existen 3 tipos de métodos:

**Estáticos:** @staticmethod, pueden ser llamadas sin tener una instancia de la clase, además este tipo de métodos, no tiene acceso al exterior, por lo cual no pueden acceder a ningún otro atributo o llamar a ninguna otra función dentro de la clase.

**Ejemplo:**

# Clase y estático

import math

class Pastel:

# Creacion del constructor y sus atributos

def \_\_init\_\_(self, ingredientes, tamaño):

self.ingredientes = ingredientes

self.tamaño = tamaño

# Como son multiples valores en un atrubito uso \_\_repr\_\_

def \_\_repr\_\_(self):

return (f'Pastel({self.ingredientes}, 'f'{self.tapaño}) !r')

def area(self):

# método para hayar el area del pasetel solo con tamaño

return self.tamaño\_area(self.tamaño)

@staticmethod

# método independiente que aisla el tamaño de los ingredientes

def tamaño\_area(A):

return A \*\* 2 \* math.pi

# Valores ingresado en clase Pastely método tamaño\_area

nuevo\_pastel = Pastel(['Harina', 'Azucar', 'Leche', 'Huevos', 'Crema'], 4)

print(nuevo\_pastel.ingredientes)

print(nuevo\_pastel.tamaño)

# Calculo realizado con el metodo statico y con valor independiente al ya ingresado

print(nuevo\_pastel.tamaño\_area(4))

**De clase:** @classmethod, comparte una característica con el método estático, la cual es que puede ser llamado sin crear una instancia de clase, lo que quiere decir que no se necesita el \_\_init\_\_ para usar este tipo de instancia de clase, este método no tiene acceso a atributos de instancias

**Ejemplo:**

class Pastel:

# Creacion del constructor y sus atributos

def \_\_init\_\_(self, ingredientes):

self.ingredientes = ingredientes

# Comprende lo que deseamos hacer con el compilador

def \_\_repr\_\_(self):

return f'pastel({self.ingredientes !r})'

# Método de clase, hacemos independiente todo lo del método

@classmethod

def Pastel\_chocolate(cls):

# Trabajamos con diferentes datos para ingredientes y que van a pertenecer a esta clase

return cls(['Harina', 'Leche', 'Chocolate'])

# Método de clase, hacemos independiente todo lo del método

@classmethod

def Pastel\_vainilla(cls):

# Trabajamos con diferentes datos para ingredientes y que van a pertenecer a esta clase

return cls(['Harina', 'Leche', 'Vainilla', 'Huevos'])

print(Pastel.Pastel\_chocolate())

print(Pastel.Pastel\_vainilla())

De instancia.

**Polimorfismo:** Clase superior define comportamientos diferentes, es la capacidad que tienen los objetos en diferentes clases para usar un comportamiento o atributo del mismo nombre, con diferente valor.

**Ejemplo:**

# Polimorfismo: mismo objeto con diferentes valores, el mismo nombre de sus atributos y métodos

# Pueden ser identicos pero sus acciones y funciones son diferentes

class Auto:

# Objeto

rueda = 4

# Accion

def desplazamiento(self):

print('El auto se está desplazando sobre 4 ruedas')

class Moto:

# Objeto

rueda = 2

# Accion

def desplazamiento(self):

print('La moto se está desplazando sobre 2 ruedas')

OTRO EJEMPLO:

# Herencia y Polimorfismo

# Clase padre

class Animales:

def \_\_init\_\_(self, nombre):

#atributos

self.nombre = nombre.upper()

# Definir a los animales por el tipo

def tipo\_animal(self):

pass

# Clase hija, como vamos a usar herencia, vamos a crear otra clase

class Leon(Animales):

def tipo\_animal(self):

print(f'{self.nombre}, es un Animal Salvaje')

# Clase hija, como vamos a usar herencia, vamos a crear otra clase

class Perro(Animales):

def tipo\_animal(self):

print(f'{self.nombre}, es un Animal Doméstico')

# Captura de datos

tipo = int(input('Seleccione el tipo de animal\n 1 León:\n 2 Perro: '))

n = input('Ingrese el nombre: ')

if tipo ==1:

# Creacion de Objetos

nuevo\_animal = Leon(n)

# Llamado del objeto

nuevo\_animal.tipo\_animal()

elif tipo == 2:

# Creacion de Objetos

nuevo\_animal = Perro(n)

# Llamado del objeto

nuevo\_animal.tipo\_animal()

else:

print(f'El tipo de animal {tipo}, no está definido')

# Polimorfismo, método y herencia

# Polimorfismo por función

class Tomate:

def tipo(self):

print('Vegetal')

def color(self):

print('Rojo')

class Manzana:

def tipo(self):

print('Fruta')

def color(self):

print('Verde')

# Funciona como funcion ya que está de manera externa a la clase

# el método, se ubica dentro de la clase

def funcion(objeto):

objeto.tipo()

objeto.color()

nuevo\_tomate = Tomate()

funcion(nuevo\_tomate)

print('\n')

nueva\_manzana = Manzana()

funcion(nueva\_manzana)

print('\n')

# Polimorfismo con métodos, funciona bien cuando tenemos más de dos clases

class Colombia:

def capital(self):

print('Santafé de Bogotá')

def idioma(self):

print('Españól')

class Francia:

def capital(self):

print('París')

def idioma(self):

print('Francés')

# Creación de los Objetos

nacionalidad = Colombia()

nacionalidad1 = Francia()

# Usado cuando existen varios objetos dentro del polimorfismo

for pais in (nacionalidad, nacionalidad1):

pais.capital()

pais.idioma()

print('\n')

# Polimorfismo con herencia

# Igual método en todas las clases, con diferente función

class Aves:

def volar(self):

print('La mayoría de las aves pueden volar, pero algunas no!')

class Aguila(Aves):

def volar(self):

print('Las aguilas, pueden volar.')

class Gallina(Aves):

def volar(self):

print('Las gallinas, no pueden volar.')

# Objetos

obj\_ave = Aves()

obj\_aguila = Aguila()

obj\_gallina = Gallina()

# Visualizacion del objeto

obj\_ave.volar()

obj\_aguila.volar()

obj\_gallina.volar()

# Encapsulamiento, es la ocultación de datos del estado interno, para proteger la integridad del objeto

# Se usa para proteger los valores de clases externas

class Cliente:

def \_\_init\_\_(self):

# Encapsular variables

self.\_\_codigo = 4321

# Método encapsulado

def \_\_cuenta(self):

print('Cuenta de Procesamiento...')

def getcodigo(self):

# Para desencapsular

return self.\_\_codigo

# Para que esta clase funcione, es necesario crear un objeto

# persona = Cliente()

# De esta forma no se puede acceder al valor cuando está encapsulado

# print(persona.\_\_codigo)

# Acceder a variables y métodos encapsulados

persona = Cliente()

# Python, protege los atributos, cambiando el nombre internamente

# para poder hacer llamados se requiere de objeto.\_nombreclase\_\_ombreatributo

print(persona.\_Cliente\_\_codigo)

# llamado del método

persona.\_Cliente\_\_cuenta()

Averiguar parámetros dinamicos