# **1. Conceptos de programación orientada a objetos: Clase y Objeto**

**Concepto de Poo:**  
Es un paradigma de programación que basa su estructura en los objetos, que son entidades o elementos que poseen datos y comportamientos, y que pertenecen a la situación que se esta analizando, teniendo una participación fundamental.

La manera en que los objetos se relacionan e interactúan entre sí, permite identificar la dinámica del contexto.

Los objetos se agrupan en clases, que son plantillas que definen los atributos y métodos que comparten los objetos de una misma clase, además de un mismo objetivo.

**Explicación2:** Es un estilo que trata los datos como objetos con atributos y métodos que pueden aplicarse a estos objetos y también ser heredados por otros objetos.

**Concepto de objeto:**Un objeto es una entidad independiente con sus propios datos y programación.

El concepto renovador de la tecnología de Orientación a Objetos es la suma de funciones a elementos de datos, a esta unión se le llama **encapsulamiento.**

**Ejemplo:**

Un objeto Auto contiene ruedas, motor, velocidad, color, etc. Llamados atributos.  
Encapsulados con estos datos se encuentran los métodos para arrancar, detenerse, doblar, frenar, etc.

La responsabilidad de un objeto consiste en realizar las acciones apropiadas y mantener actualizados sus datos internos.

Cuando otra parte del programa (otros objetos) necesitan que el auto realice alguna de estas tareas (por ejemplo, arrancar) le envía un mensaje. A estos objetos que envían mensajes no les interesa la manera en que el objeto auto lleva a cabo sus tareas ni las estructuras de datos que maneja, por ello, están ocultos.

Entonces, un objeto contiene información pública, lo que necesitan los otros objetos para interactuar con el e información privada, interna, lo que necesita el objeto para operar y que es irrelevante para los otros objetos de la aplicación.

# **2. Declaración de una clase y creación de objetos**

Poo se basa en la definición de clases; a diferencia de la programación estructurada, que esta centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Un poco mas abajo se define una clase Persona y luego se crean dos objetos de dicha clase.

Una clase es una plantilla (molde), que define atributos (lo que conocemos como variables) y métodos (lo que conocemos como funciones).

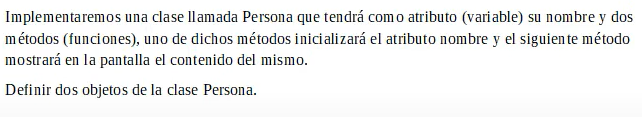
Debemos declarar una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto apropiadamente dicho.

**Self:**

Referencia del objeto que llama al método. Se utiliza para hacer referencia a la instancia actual de una clase. En este contexto, se refiere al objeto en si mismo que esta siendo manipulado o utilizado en un programa.

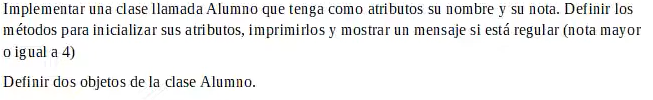
**Explicación:**Se agrega self para que pueda acceder a las variables de instancia “Nombre” y “edad”

**Problema 1**

****

class Persona:  
 def inicializar(self,nombre):  
 self.nombre = nombre  
 def Imprimir(self):  
 print(f"Nombre: {self.nombre}")  
  
persona1 = Persona()  
persona1.inicializar("Richard")  
persona1.Imprimir()  
  
persona2 = Persona()  
persona2.inicializar("Ana")  
persona2.Imprimir()

**Problema 2**

****

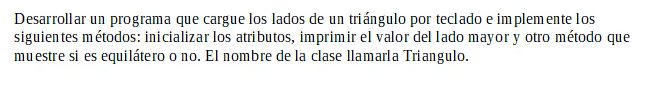
class Alumno:  
 def inicializar(self,nombre,nota):  
 self.nombre = nombre  
 self.nota = nota  
 def imprimir(self):  
 print("Nombre: ",self.nombre)  
 print("nota: ",self.nota)  
  
 def estado(self):  
 if self.nota >=4:  
 print(self.nombre,"esta regular")  
 else:  
 print(self.nombre,"esta libre")  
alumno1 = Alumno()  
alumno1.inicializar("juan",2)  
alumno1.imprimir()  
alumno1.estado()  
  
alumno2 = Alumno()  
alumno2.inicializar("ana",10)  
alumno2.imprimir()  
alumno2.estado()

**Problema 3**

****

class Persona:  
 def inicializar(self,nombre,edad):  
 self.nombre = nombre  
 self.edad  
 def imprimir(self):  
 print("nombre: ",self.nombre)  
 print("edad: ",self.edad)  
 def MayorEdad(self):  
 if self.edad >=18:  
 print("Es mayor de edad")  
 else:  
 print("Es menor de edad")  
persona1 = Persona()  
persona1.inicializar("Luis",23)  
persona1.imprimir()

**Problema 4:**

****

class Triangulo:  
  
 def inicializar(self):  
 self.lado1 = int(input("Ingrese lado 1: "))  
 self.lado2 = int(input("Ingrese lado 2: "))  
 self.lado3 =int(input("Ingrese lado 3: "))  
 def imprimir(self):  
 print("lado 1: ",self.lado1)  
 print("lado 2: ",self.lado2)  
 print("lado 3: ",self.lado3)  
 def lado\_mayor(self):  
 if self.lado1>self.lado2 and self.lado1>self.lado3:  
 print(self.lado1)  
 elif self.lado2>self.lado1 and self.lado2>self.lado3:  
 print(self.lado2)  
 else:  
 print(self.lado3)  
 def equilatero(self):  
 if self.lado1 == self.lado2 and self.lado1 == self.lado3:  
 print("El triangulo es equilatero")  
 else:  
 print("El triangulo no es equilatero")  
triangulo1 = Triangulo()  
triangulo1.inicializar()  
triangulo1.imprimir()  
triangulo1.lado\_mayor()  
triangulo1.equilatero()

**Método \_\_init\_\_ de la clase**

El método \_\_init\_\_ es un método especial de una clase en Python. El objetivo fundamental del método \_\_init\_\_ es inicializar los atributos del objeto que creamos.

Básicamente el método \_\_init\_\_ reemplaza al método inicializar que habíamos hecho en el concepto anterior.

Las ventajas de implementar el método \_\_init\_\_ en lugar del método inicializar son:

1. El método \_\_init\_\_ es el primer método que se ejecuta cuando se crea un objeto
2. El método \_\_init\_\_ se llama automáticamente. Es decir es imposible olvidarse de llamarlo ya que se llamara automáticamente.
3. Quien utiliza POO en Python conoce el objetivo de este método.

Otra característica del método \_\_init\_\_son:

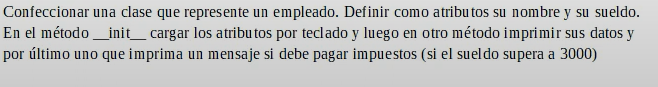
* Se ejecuta inmediatamente luego de crear un objeto.
* El método init no puede retornar datos.
* El método init no puede recibir parámetros que se utilizan normalmente para inicializar atributos.
* El método init es un método opcional, de todos modos, es muy común declararlo.

Veamos la sintaxis del constructor:

Def \_\_init\_\_([parámetros]):

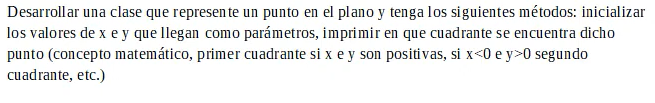
[algoritmo]

**Problema 1:**

****

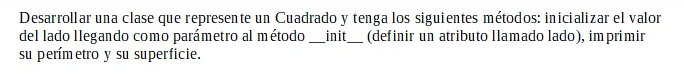
class Empleado:  
 def \_\_init\_\_(self):  
 self.nombre = input("nombre empleado: ")  
 self.sueldo = float(input("sueldo empleado: "))  
 def imprimir(self):  
 print("nombre: ",self.nombre)  
 print("sueldo: ",self.sueldo)  
 def pagaimpuestos(self):  
 if self.sueldo >3000:  
 print("debe pagar impuestos")  
 else:  
 print("no paga impuestos")  
empleado1 = Empleado()  
empleado1.imprimir()  
empleado1.pagaimpuestos()

**Problema 2:**

****

class Punto:  
 def \_\_init\_\_(self,x=None,y=None):  
 self.x = x  
 self.y = y  
 def imprimir(self):  
 print("Coordenada del punto: ")  
 print(f"({self.x},{self.y})")  
 def imprimir\_cuadrante(self):  
 if self.x>0 and self.y>0:  
 print("primer cuadrante")  
 elif self.x<0 and self.y>0:  
 print("segundo cuadrante")  
 elif self.x<0 and self.y<0:  
 print("tercer cuadrante")  
 elif self.x>0 and self.y<0:  
 print("cuarto cuadrante")  
 else:  
 print("esta en un eje")  
  
punto1 = Punto()  
punto1.imprimir()  
punto1.imprimir\_cuadrante()

**Problema 3**

****

class Cuadrado:  
 def \_\_init\_\_(self,lado):  
 self.lado = lado  
 def mostrar\_perimetro(self):  
 perimetro = self.lado \* 4  
 print(f"el perimetro del cuadrado es: {perimetro}")  
 def mostrar\_superficie(self):  
 superficie = self.lado \* self.lado  
 print("La superficie del cuadrado es: ",superficie)  
  
#perimetro no le agregamos el indicador self porque no lo queremos definir como un atributo de la clase  
#es una variable local al metodo "mostrar\_perimetro"  
#en otro metodo a esa variable perimetro no lo puedo acceder  
#pero si se puede acceder en otro metodo al atributo lado, ya que le antecedemos el indicador self

**Problema 4**

****

class Operaciones:  
 def \_\_init\_\_(self):  
 self.valor1 = int(input("Ingrese un valor 1: "))  
 self.valor2 = int(input("Ingrese un valor 2: "))  
 def sumar(self):  
 suma = self.valor1 + self.valor2  
 print("la suma de los 2 valores es: ",suma)  
 def restar(self):  
 resta = self.valor1 - self.valor2  
 print("la resta es: ",resta)  
 def multiplicar(self):  
 mult = self.valor1 \* self.valor2  
 print("la multiplicacion es: ",mult)  
 def dividir(self):  
 try:  
 div = self.valor1 / self.valor2  
 print("la division es: ",div)  
 except ZeroDivisionError:  
 print("No es posible dividir por 0")

**Llamada de métodos desde otro método de la misma clase**

Hasta ahora todos los problemas planteados hemos llamado a los métodos desde donde definimos un objeto de dicha clase, por ejemplo:

Empleado1 = Empleado(“diego”,2000)

Empleado1.paga\_impuestos()

Utilizamos la sintaxis:

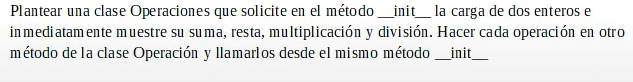
[nombre del objeto].[nombre del método]

Es decir antecedemos al nombre del método el nombre del objeto y el operador punto.  
Ahora bien que pasa si queremos llamar dentro de la clase a otro método que pertenece a la misma clase, la sintaxis es la siguiente:

Self.[nombre del método]

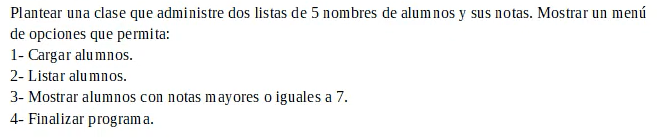
Es importante tener en cuenta que esto solo se puede hacer cuando estamos dentro de la mima clase.

**Problema 1**

****

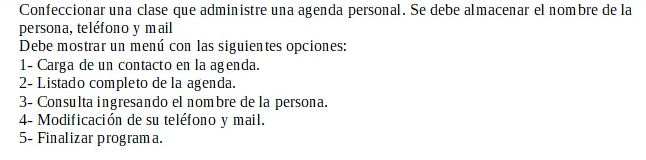
class Operaciones:  
 def \_\_init\_\_(self):  
 self.valor1 = int(input("valor 1: "))  
 self.valor2 = int(input("valor 2: "))  
 self.sumar()  
 self.restar()  
 self.multiplicar()  
 self.division()  
 def sumar(self):  
 suma = self.valor1 + self.valor2  
 print("La suma es: ",suma)  
 def restar(self):  
 resta = self.valor1 - self.valor2  
 print("la resta es: ",resta)  
 def multiplicar(self):  
 multi = self.valor1 \* self.valor2  
 print("la multiplicacion es: ",multi)  
 def division(self):  
 div = self.valor1 / self.valor2  
 print("la division es: ",div)  
operaciones1 = Operaciones()

**Problema 2**

****

class Alumno:  
 def \_\_init\_\_(self):  
 self.nombres = []  
 self.notas = []  
 def menu(self):  
 opcion = 0  
 while opcion!=4:  
 print("1 - Carga de alumnos")  
 print("2 - Listado de alumnos")  
 print("3 - Listado de alumnos con notas mayores o iguales a 7")  
 print("4 Finalizar el programa")  
 opcion = int(input("Ingrese su opcion: "))  
 if opcion == 1:  
 self.cargar()  
 elif opcion == 2:  
 self.listar()  
 elif opcion == 3:  
 self.nota\_superior()  
 def cargar(self):  
 for x in range(5):  
 nombre = input("Ingrese el nombre del alumno: ")  
 nota = int(input("Ingrese la nota: "))  
 self.nombres.append(nombre)  
 self.notas.append(nota)  
 def listar(self):  
 for x,valor in enumerate(self.nombres):  
 print(f"registro nro: {x+1}: nombre: {valor},nota: {self.notas[x]}")  
 def nota\_superior(self):  
 print("Notas iguales o superiores a 7")  
 for x, valor in enumerate(self.notas):  
 if valor >=7:  
 print(f"registro nro {x+1}, nombre: {self.nombres[x]}, nota: {valor}")  
  
alumnos1 = Alumno()  
alumnos1.menu()

**Problema 3**

****

class Agenda:  
 def \_\_init\_\_(self):  
 self.contactos = {}  
 def menu(self):  
 opcion = 0  
 while opcion!=5:  
 print("1. Carga de un contacto en la agenda")  
 print("2. Listado completo de la agenda")  
 print("3. Consulta ingresando el nombre de una persona")  
 print("4. Modificaciones del telefono y mail")  
 print("5. Finalizar el programa")  
 opcion = int(input("Ingrese la opcion: "))  
 if opcion == 1:  
 self.cargar()  
 elif opcion == 2:  
 self.listado()  
 elif opcion == 3:  
 self.consulta()  
 elif opcion == 4:  
 self.modificacion()  
 def cargar(self):  
 nombre = input("Nombre: ")  
 telefono = int(input("Telefono: "))  
 mail = input("Mail: ")  
 self.contactos[nombre]=(telefono,mail)  
 print()  
 def listado(self):  
 for nombre, (telefono, mail) in self.contactos.items():  
 print(f"nombre: {nombre}, telefono: {telefono}, mail: {mail}")  
 def consulta(self):  
 nombref = input("Ingrese el nombre de la persona a consultar: ")  
 if nombref in self.contactos:  
 print(f"Su telefono es: {self.contactos[nombref][0]}, y su mail es: {self.contactos[nombref][1]}")  
 def modificacion(self):  
 nombre = input("Ingrese el nombre de la persona a modificar: ")  
 if (nombre in self.contactos):  
 telefono = int(input("Ingrese nuevo contacto: "))  
 mail = input("Ingrese nuevo mail: ")  
 self.contactos[nombre]=(telefono,mail)  
 else:  
 print("No existe el nombre")  
agenda1 = Agenda()  
agenda1.menu()

**Colaboración de clases**

Normalmente un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.

Planteamos un problema separando las actividades en dos clases.

**Problema:**

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcula la cantidad de dinero que hay depositado.

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Cliente y la clase Banco:

Luego debemos definir los atributos y los métodos de cada clase:

Cliente:

Atributos:

Nombre

Monto

Métodos:

\_\_init\_\_

Depositar

Extraer

Retornar\_monto

Imprimir

Banco:

Atributos:

3 cliente (3 objeto de la clase cliente)

Métodos:

\_\_init\_\_

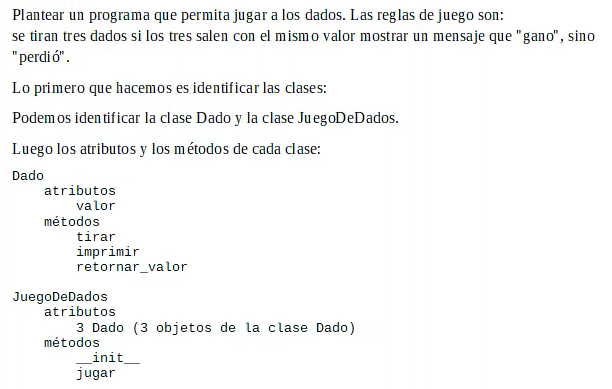
Operar

Depósitos\_totales

**Problema 1**

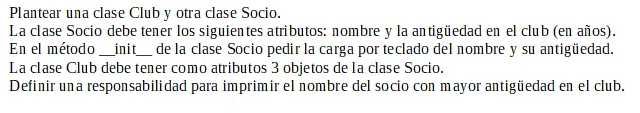
class Cliente:  
 def \_\_init\_\_(self,nombre):  
 self.nombre = nombre  
 self.monto = 0  
 def depositar(self,monto):  
 self.monto = self.monto+monto  
 def extraer(self,monto):  
 self.monto = self.monto - monto  
 def retornar\_monto(self):  
 return self.monto  
 def imprimir(self):  
 print(self.nombre,"tiene depositado la suma de",self.monto)  
class Banco:  
 def \_\_init\_\_(self):  
 self.cliente1 = Cliente("Juan")  
 self.cliente2 = Cliente("Ana")  
 self.cliente3 = Cliente("Luis")  
 def operar(self):  
 self.cliente1.depositar(100)  
 self.cliente2.depositar(100)  
 self.cliente3.depositar(200)  
 self.cliente3.extraer(100)  
 def depositos\_totales(self):  
 total = self.cliente1.retornar\_monto() + self.cliente2.retornar\_monto() + self.cliente3.retornar\_monto()  
 print("Total de dinero del banco",total)  
 self.cliente1.imprimir()  
 self.cliente2.imprimir()  
 self.cliente3.imprimir()  
banco1 = Banco()  
banco1.operar()  
banco1.depositos\_totales()

**Problema 2:**

****

import random  
class Dado:  
 def tirar(self):  
 self.valor = random.randint(1,6)  
 def imprimir(self):  
 print("Valor del dado: ",self.valor)  
 def retornar\_valor(self):  
 return self.valor  
class JuegoDeDados:  
 def \_\_init\_\_(self):  
 self.dado1 = Dado()  
 self.dado2 = Dado()  
 self.dado3 = Dado()  
 def jugar(self):  
 self.dado1.tirar()  
 self.dado1.imprimir()  
 self.dado2.tirar()  
 self.dado2.imprimir()  
 self.dado3.tirar()  
 self.dado3.imprimir()  
 if (self.dado1.retornar\_valor()== self.dado2.retornar\_valor()  
 and self.dado1.retornar\_valor() == self.dado3.retornar\_valor()):  
 print("gano")  
 else:  
 print("perdio")  
juego1 = JuegoDeDados()  
juego1.jugar()

**Problema 3:**

****

class Socio:  
  
 def \_\_init\_\_(self):  
 self.nombre = input("Ingrese nombre de socio: ")  
 self.antiguedad = int(input("Ingrese la antiguedad: "))  
 def imprimir(self):  
 print(self.nombre,"tiene una antiguedad de",self.antiguedad)  
 def retornar\_antiguedad(self):  
 return self.antiguedad  
class Club:  
 def \_\_init\_\_(self):  
 self.socio1 = Socio()  
 self.socio2 = Socio()  
 self.socio3 = Socio()  
 def mayor\_antiguedad(self):  
 print("Socio con mayor antiguedad")  
 if self.socio1.retornar\_antiguedad()>self.socio2.retornar\_antiguedad() and self.socio1.retornar\_antiguedad()>self.socio3.retornar\_antiguedad():  
 self.socio1.imprimir()  
 elif self.socio2.retornar\_antiguedad()>self.socio1.retornar\_antiguedad() and self.socio2.retornar\_antiguedad()>self.socio3.retornar\_antiguedad():  
 self.socio2.imprimir()  
 else:  
 self.socio3.imprimir()  
club1 = Club()  
club1.mayor\_antiguedad()

# **3. Herencia**

Vimos en el concepto anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relaciones entre clases que es la herencia.

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todos los atributos y métodos de su “super clase” o “clase padre” y además se le podrán añadir otro atributos y métodos propios.

**Clase padre**

Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la clase padre.

**Subclase**

Clase descendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase.

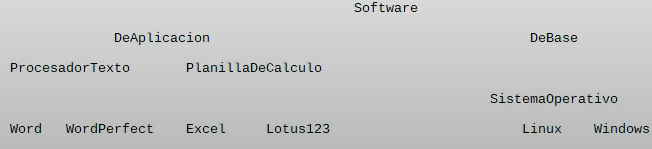
Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

Veamos algunos ejemplos teóricos de herencia:

1. Imaginemos la clase Vehículo. ¿Qué clases podrían derivar de ella?



1. ¿Imaginemos la clase Software? ¿Qué clases podrían derivar de ella?

****

El primer tipo de relación que habíamos visto entre dos clases es la colaboración. Recordemos que es cuando una clase contiene un objeto de otra clase como atributo.

Cuando la relación entre dos clases es del tipo “…tienen un…” o “…es parte de…”, no debemos implemente herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe una relación de tipo “…tiene un…”,no debe implementarse herencia sino declarar en la clase ClaseA un atributo de la ClaseB.

Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto “..tiene 4…” “Rueda, Volante”…es parte de…”Auto: pero la clase Auto no debe derivar de Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración.

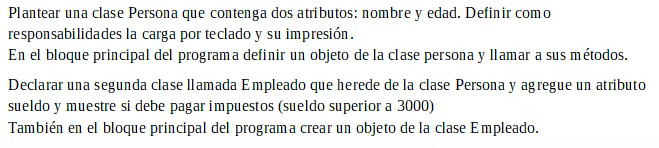
Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.

Luego si vemos que dos clase responden a la pregunta ClaseA “…es una …” ClaseB es posible que haya una relación de herencia.

**Por ejemplo:**

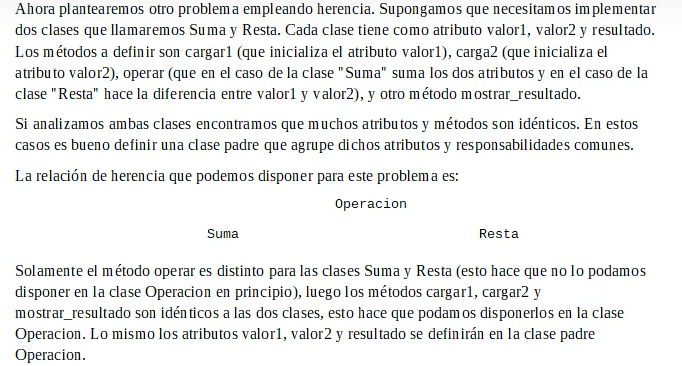
Auto “es un” Vehículo  
Circulo “es una” figura  
Mouse “es un” DispositivoEntrada  
Suma “es una” Operación

**Problema 1**

****

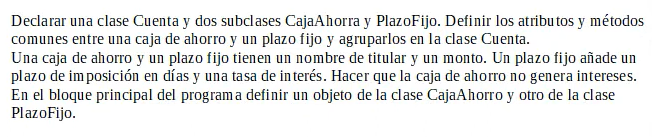
class Persona:  
 def \_\_init\_\_(self):  
 self.nombre = input("Ingrese nombre: ")  
 self.edad = int(input("Ingrese la edad: "))  
 def imprimir(self):  
 print(f"Nombre: {self.nombre}")  
 print(f"Edad: {self.edad}")  
class Empleado(Persona):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.sueldo = float(input("Ingrese sueldo"))  
 def imprimir(self):  
 super().imprimir()  
 print(f"Sueldo: {self.sueldo}")  
 def paga\_impuestos(self):  
 if self.sueldo>3000:  
 print(self.nombre,"debe pagar impuestos")  
 else:  
 print(self.nombre,"no paga impuestos")  
  
  
empleado1 = Empleado()  
empleado1.imprimir()  
empleado1.paga\_impuestos()

**Problema 2**

****

class Operacion:  
 def \_\_init\_\_(self):  
 self.valor1 = 0  
 self.valor2 = 0  
 self.resultado = 0  
 def cargar1(self):  
 self.valor1 = int(input("Ingrese el primer valor: "))  
 def cargar2(self):  
 self.valor2 = int(input("Ingrese el segundo valor: "))  
 def mostrar\_resultado(self):  
 print(self.resultado)  
 def operar(self):  
 pass  
class Suma(Operacion):  
 def operar(self):  
 self.resultado = self.valor1 + self.valor2  
class Resta(Operacion):  
 def operar(self):  
 self.resultado = self.valor1 - self.valor2  
suma1 = Suma()  
suma1.cargar1()  
suma1.cargar2()  
suma1.operar()  
suma1.mostrar\_resultado()  
  
resta1 = Resta()  
resta1.cargar1()  
resta1.cargar2()  
resta1.operar()  
resta1.mostrar\_resultado()

**Problema 3**

****

class Cuenta:  
 def \_\_init\_\_(self,titular,monto):  
 self.titular = titular  
 self.monto = monto  
 def imprimir(self):  
 print("Titular: ",self.titular)  
 print("Monto: ",self.monto)  
class CajaAhorro(Cuenta):  
 def \_\_init\_\_(self,titular,monto):  
 super().\_\_init\_\_(titular,monto)  
 def imprimir(self):  
 print("Cuenta de caja de ahorro")  
 super().imprimir()  
class PlazoFijo(Cuenta):  
 def \_\_init\_\_(self,titular,monto,plazo,interes):  
 super().\_\_init\_\_(titular,monto)  
 self.plazo = plazo  
 self.interes = interes  
 def imprimir(self):  
 print("Cuenta de plazo fijo")  
 super().imprimir()  
 print("Plazo en dias: ",self.plazo)  
 print("Intereses: ",self.interes)  
 self.ganancia()  
 def ganancia(self):  
 gan = self.monto \* self.interes / 100  
 print(f"monto de interes {gan}")  
  
cajadeahorro1 = CajaAhorro("Juan",1300)  
cajadeahorro1.imprimir()  
  
plazofijo1 = PlazoFijo("Ana",1000,30,0.75)  
plazofijo1.imprimir()

# **4. Variables de clase**

Hemos visto como definimos atributos en una clase anteponiendo la palabra clase self:

Class Persona:

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

Self.nombre = nombre

Los atributos son independientes por cada objeto o instancia de la clase, es decir si definimos tres objetos de la clase Persona, todas las personas tienen un atributo nombre pero cada uno tiene un valor independiente:

Class Persona:

Def \_\_ini\_\_(self,nombre):

Self.nombre = nombre

#Bloque principal

Persona1 = Persona(“Juan”)

Persona2 = Persona(“Ana”)

Persona3 = Persona(“Luis”)

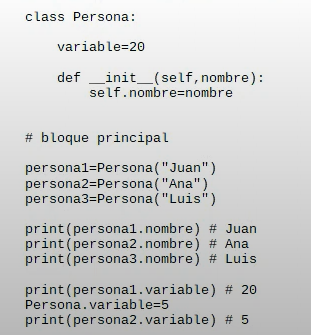
Print(persona1.nombre)

Print(persona2.nombre)

Print(persona3.nombre)

En algunas situaciones necesitamos almacenar datos que sean compartidos por todos los objetos de dicha clase, en esas situaciones debemos emplear variables de clase.

Para definir una variable de clase lo hacemos dentro de la clase, pero fuera de sus métodos.

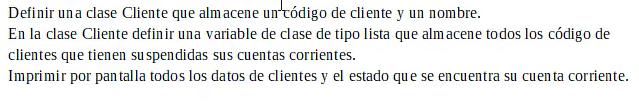


Se reserva solo un espacio para la variable “variable”, independientemente que se definan muchos objetos de la clase Persona. La variable “variable” es compartida por todos los objetos persona1, persona2 y persona3.

Para modificar la variable de clase hacemos referencia al nombre de la clase y seguidamente el nombre de la variable.

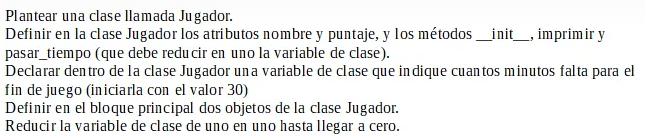
Persona.variable = 5

**Problema 1**

****

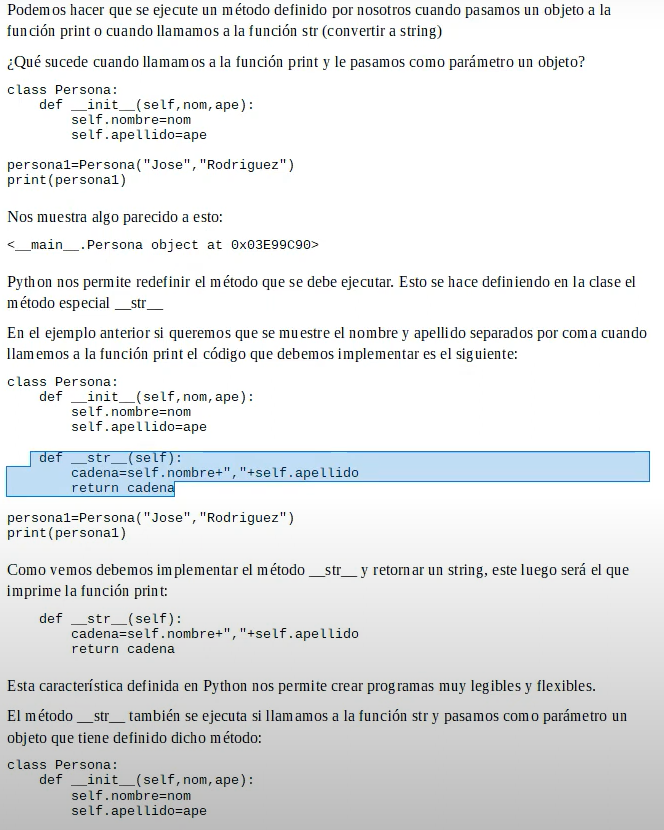
class Cliente:  
 suspendidos = []  
 def \_\_init\_\_(self,codigo,nombre):  
 self.codigo = codigo  
 self.nombre = nombre  
 def imprimir(self):  
 print("Codigo: ",self.codigo)  
 print("Nombre: ",self.nombre)  
 self.esta\_suspendido()  
 def esta\_suspendido(self):  
 if self.codigo in Cliente.suspendidos:  
 print("Esta suspendido")  
 else:  
 print("No esta suspendido")  
 print()  
 def suspender(self):  
 Cliente.suspendidos.append(self.codigo)  
cliente1 = Cliente(1234,"richard")  
cliente2 = Cliente(2121,"carlos")  
cliente3 = Cliente(2134,"juan")  
cliente4 = Cliente(1311,"pedro")  
  
cliente2.suspender()  
cliente3.suspender()  
  
cliente1.imprimir()  
cliente2.imprimir()  
cliente3.imprimir()  
cliente4.imprimir()  
  
print(Cliente.suspendidos)

**Problema 2**

****

class Jugador:  
 tiempo = 30  
 def \_\_init\_\_(self,nombre,puntaje):  
 self.nombre = nombre  
 self.puntaje = puntaje  
 def imprimir(self):  
 print("Noombre: ",self.nombre)  
 print("Puntaje: ",self.puntaje)  
 print("Fin de juego en ",Jugador.tiempo)  
 def pasar\_minuto(self):  
 Jugador.tiempo-=1  
jugador1 = Jugador("richard",100)  
jugador2 = Jugador("Juan",50)  
while Jugador.tiempo>0:  
 jugador1.imprimir()  
 jugador2.imprimir()  
 jugador1.pasar\_minuto()

**Método especial \_\_str\_\_**



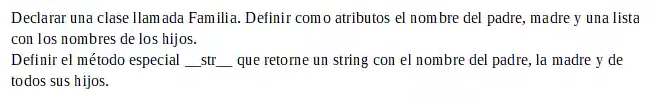


**Problema 1**

****

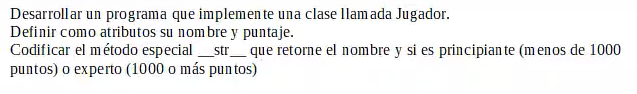
class Punto:  
 def \_\_init\_\_(self,x,y):  
 self.x = x  
 self.y = y  
 def \_\_str\_\_(self):  
 return f"({self.x}),({self.y})"  
punto1 = Punto(21,23)  
punto2 = Punto(-21,3)  
print(str(punto1))

**Problema 2**

****

class Familia:  
 def \_\_init\_\_(self,padre,madre,hijos = ["No tiene hijos"]):  
 self.padre = padre  
 self.madre = madre  
 self.hijos = hijos  
 def \_\_str\_\_(self):  
 cadena = f"Nombre del padre: {self.padre}, Nombre de la madre: {self.madre}, Nombre de los hijos: "  
 if self.hijos != ["No tiene hijos"]:  
 cadena+=", ".join(self.hijos)  
 return cadena  
 else:  
 cadena+=", ".join(self.hijos)  
 return cadena  
familia1 = Familia("Pablo","Ana",["Carlos","Veronica"])  
familia2 = Familia("Jorge","Carla")  
familia3 = Familia("Luis","Maria",["Marcos"])  
  
print(familia1)  
print(familia2)  
print(familia3)

**Problema 3**

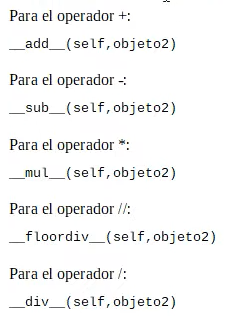
****

class Jugador:  
 def \_\_init\_\_(self,nombre,puntaje):  
 self.nombre = nombre  
 self.puntaje = puntaje  
 def \_\_str\_\_(self):  
 cadena = f"Nombre del jugador: {self.nombre},rango del jugador: "  
 if self.puntaje<1000:  
 cadena+="Principiante"  
 return cadena  
 else:  
 cadena+="Experto"  
 return cadena  
jugador1 = Jugador("Ana",700)  
jugador2 = Jugador("Juan",2000)  
  
print(jugador1)  
print(jugador2)

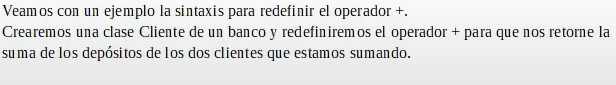
# **5. Redefinición de los operadores matemáticos con objetos**

Python nos permite redefinir los operadores matemáticos cuando planteamos una clase.

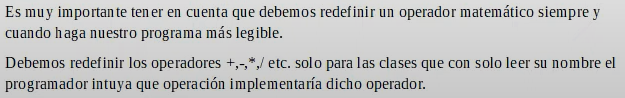
Los métodos especiales que debemos implementar son los siguientes:



**Problema1:**

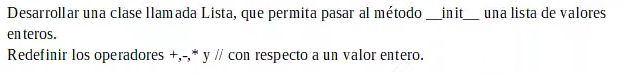
****

**Acotaciones:**

****

class Cliente:  
 def \_\_init\_\_(self,nombre,monto):  
 self.nombre = nombre  
 self.monto = monto  
 def \_\_add\_\_(self, objeto2):  
 suma = self.monto + objeto2.monto  
 return suma  
cliente1 = Cliente("Ana",1200)  
cliente2 = Cliente("Juan",1000)  
print("Total depositado por los dos clientes")  
print(cliente1+cliente2)

**Problema 2**

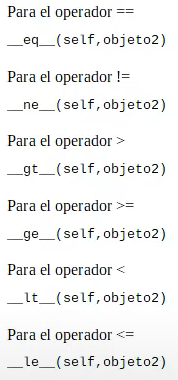
****

class Lista:  
 def \_\_init\_\_(self,lista):  
 self.lista = lista  
 def imprimir(self):  
 print(self.lista)  
 def \_\_add\_\_(self, other):  
 nueva = []  
 for i, valor in enumerate(self.lista):  
 nueva.append(valor + other)  
 return nueva  
 def \_\_sub\_\_(self, other):  
 nueva = []  
 for i, valor in enumerate(self.lista):  
 nueva.append(valor - other)  
 return nueva  
 def \_\_mul\_\_(self, other):  
 nueva = []  
 for i, valor in enumerate(self.lista):  
 nueva.append(valor \* other)  
 return nueva  
 def \_\_floordiv\_\_(self, other):  
 nueva = []  
 for i, valor in enumerate(self.lista):  
 nueva.append(valor // other)  
 return nueva  
lista1 = Lista([2,2,2,2,2])  
lista1.imprimir()  
print(lista1 // 2)

# **6. Redefinición de los operadores relacionales con objetos**

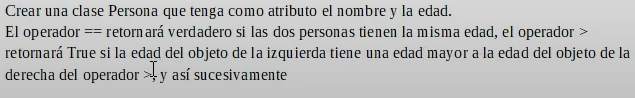
Python también nos permite redefinir los operadores relacionales cuando planteamos una clase.

Los métodos especiales que podemos implementar son los siguientes:



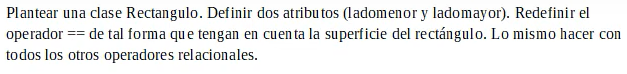
Es importante recordar que una redefinición de un operador tiene sentido si ayuda y hace más claro nuestro algoritmo.

**Problema 1**



class Persona:  
 def \_\_init\_\_(self,nombre,edad):  
 self.nombre = nombre  
 self.edad = edad  
 def \_\_eq\_\_(self, other):  
 if self.edad == other.edad:  
 return True  
 else:  
 return False  
 def \_\_ne\_\_(self, other):  
 if self.edad != other.edad:  
 return True  
 else:  
 return False  
 def \_\_gt\_\_(self, other):  
 if self.edad > other.edad:  
 return True  
 else:  
 return False  
 def \_\_ge\_\_(self, other):  
 if self.edad >= other.edad:  
 return True  
 else:  
 return False  
 def \_\_lt\_\_(self, other):  
 if self.edad < other.edad:  
 return True  
 else:  
 return False  
 def \_\_le\_\_(self, other):  
 if self.edad <= other.edad:  
 return True  
 else:  
 return False  
  
persona1 = Persona("Richard",24)  
persona2 = Persona("Juan",22)  
def funcion():  
 if persona1 == persona2:  
 print("Las dos personas tienen la misma edad")  
 else:  
 print("No tienen la misma edad")  
def funcion1():  
 if persona1 != persona2:  
 print("Las dos personas no tienen la misma edad")  
def funcion2():  
 if persona1 > persona2:  
 print("Richard es mayor que juan")  
def funcion3():  
 if persona1 >= persona2:  
 print("Richard es mayor o igual que juan")  
def funcion4():  
 if persona1 < persona2:  
 print("Richard es menor que juan")  
 else:  
 print("Juan es menor que Richard")  
def funcion5():  
 if persona1 <= persona2:  
 print("Richard es menor o igual que juan")  
 else:  
 print("Juan es menor o igual que Richard")  
  
funcion2()

**Problema 2**

****

class Rectangulo:  
 def \_\_init\_\_(self,ladomenor,ladomayor):  
 self.ladomenor = ladomenor  
 self.ladomayor = ladomayor  
 def retornar\_superficie(self):  
 return self.ladomenor \* self.ladomayor  
 def \_\_eq\_\_(self, other):  
 if self.retornar\_superficie() == other.retornar\_superficie():  
 return True  
 else:  
 return False  
 def \_\_ne\_\_(self, other):  
 if self.retornar\_superficie() != other.retornar\_superficie():  
 return True  
 else:  
 return False  
 def \_\_gt\_\_(self, other):  
 if self.retornar\_superficie() > other.retornar\_superficie():  
 return True  
 else:  
 return False  
 def \_\_ge\_\_(self, other):  
 if self.retornar\_superficie() >= other.retornar\_superficie():  
 return True  
 else:  
 return False  
 def \_\_lt\_\_(self, other):  
 if self.retornar\_superficie() < other.retornar\_superficie():  
 return True  
 else:  
 return False  
 def \_\_le\_\_(self, other):  
 if self.retornar\_superficie() <= other.retornar\_superficie():  
 return True  
 else:  
 return False  
  
rectangulo1 = Rectangulo(11,10)  
rectangulo2 = Rectangulo(5,20)  
  
def funcion6():  
 if rectangulo1 < rectangulo2:  
 print("rectangulo 1 es menor o igual que rectangulo 2")  
def funcion5():  
 if rectangulo1 < rectangulo2:  
 print("rectangulo 1 es menor que rectangulo 2")  
def funcion4():  
 if rectangulo1 >= rectangulo2:  
 print("rectangulo 1 es mayor o igual que rectangulo 2")  
def funcion1():  
 if rectangulo1 == rectangulo2:  
 print("Los rectangulos tienen la misma superficie")  
def funcion2():  
 if rectangulo1 != rectangulo2:  
 print("No tienen la misma superficie")  
def funcion3():  
 if rectangulo1 > rectangulo2:  
 print("rectangulo 1 es mayor que rectangulo 2")  
  
funcion4()