1-Implementar una clase llamada Alumno que tenga como atributos su nombre y su nota. Definir los métodos para inicializar sus atributos, imprimirlos y mostrar un mensaje si está regular (nota mayor o igual a 4) Definir dos objetos de la clase Alumno.

```
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 mostrar_estado(self):
        if self.nota>=4:
            print("Regular")
        else:
            print("Libre")

# bloque principal

alumno1=Alumno()
alumno1.inicializar("diego",2)
alumno1.imprimir()
alumno1.mostrar_estado()

alumno2=Alumno()
alumno2.inicializar("ana",10)
alumno2.imprimir()
alumno2.mostrar_estado()
```

diferenciamos los atributos del objeto antecediendo el identificador self

cuando se crean los atributos en el método inicializar luego podemos acceder a los mismos en los otros métodos de la clase, por ejemplo en el método mostrar_estado verificamos el valor almacenado en el atributo nota

2-Confeccionar una clase que permita carga el nombre y la edad de una persona. Mostrar los datos cargados. Imprimir un mensaje si es mayor de edad (edad>=18)
Definir dos objetos de la clase Persona.

```
class Persona:

    def inicializar(self,nombre,edad):
        self.nombre=nombre
        self.edad=edad

    def imprimir(self):
        print("Nombre",self.nombre)
        print("Edad",self.edad)

    def mayor_edad(self):
        if self.edad>=18:
            print("Es mayor de edad")
        else:
            print("No es mayor de edad")

# bloque principal

personal=Persona()
personal.inicializar("diego",40)
personal.imprimir()
personal.mayor_edad()
```

3- Confeccionar una clase que represente un empleado. Definir como atributos su nombre y su sueldo. En el método __init__ cargar los atributos por teclado y luego en otro método imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000)

Definimos el método __init__ donde cargamos por teclado el nombre del empleado y su sueldo. Inmediatamente creamos el objeto se ejecuta no es necesario llamarlo

```
class Empleado:

    def __init__(self):
        self.nombre=input("Ingrese el nombre del empleado:")
        self.sueldo=float(input("Ingrese el sueldo:"))

    def imprimir(self):
        print("Nombre:",self.nombre)
        print("Sueldo:",self.sueldo)

    def paga_impuestos(self):
        if self.sueldo>3000:
            print("Debe pagar impuestos")
        else:
            print("No paga impuestos")

# bloque principal
empleado1=Empleado()
empleado1.imprimir()
empleado1.paga_impuestos()
```

```
>>>
======= RESTART: F:/TerciarioUquiza/I
Ingrese el nombre del empleado:pepe
Ingrese el sueldo:2000
Nombre: pepe
Sueldo: 2000.0
No paga impuestos
>>>
```

4-Implementar la clase Operaciones. Se deben cargar dos valores enteros por teclado en el método __init__, calcular su suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

```
class Operaciones:

    def __init__(self):
        self.valor1=int(input("Ingrese primer valor:"))
        self.valor2=int(input("Ingrese segundo valor:"))

    def sumar(self):
        su=self.valor1+self.valor2
        print("La suma es",su)

    def restar(self):
        re=self.valor1-self.valor2
        print("La resta es",re)

    def multiplicar(self):
        pro=self.valor1*self.valor2
        print("El producto es",pro)

    def division(self):
        divi=self.valor1/self.valor2
        print("La division es",divi)

# bloque principal
operacion1=Operaciones()
operacion1.sumar()
operacion1.restar()
operacion1.multiplicar()
operacion1.division()
```

```
>>>
======= RESTART: F:/TerciarioUqui
Ingrese primer valor:120
Ingrese segundo valor:110
La suma es 230
La resta es 10
El producto es 13200
La division es 1.0909090909090908
>>>
```

Llamada de métodos desde otro método de la misma clase

```
class Operacion:

    def __init__(self):
        self.valor1=int(input("Ingrese primer valor:"))
        self.valor2=int(input("Ingrese segundo valor:"))
        self.sumar()
        self.restar()
        self.multiplicar()
        self.dividir()

    def sumar(self):
        suma=self.valor1+self.valor2
        print("La suma es",suma)

    def restar(self):
        resta=self.valor1-self.valor2
        print("La rersta es",resta)

    def multiplicar(self):
        multi=self.valor1*self.valor2
        print("El producto es",multi)

    def dividir(self):
        divi=self.valor1/self.valor2
        print("La division es",divi)


# bloque principal
operacion1=Operacion()
```

Plantear una clase Operaciones que solicite en el método __init__ la carga de dos enteros e inmediatamente muestre su suma, resta, multiplicación y división. Hacer cada operación en otro método de la clase Operación y llamarlos desde el mismo método __init__

La clase anterior realizamos este ejercicio pero llamando a los métodos desde el objeto en forma independiente. Ahora lo haremos desde el método __init__

Colaboración de clases

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

Cuando una **clase contiene un objeto de otra clase como atributo**.
Cuando la relación entre dos clases es del tipo "...tiene un..." o "...es parte de...", estamos frente a una relación de **colaboración de clases**

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule 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
Banco
    atributos
        3 Cliente (3 objetos de la clase Cliente)
    métodos
        __init__
        operar
        depositos_totales
```

```
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("Diego")

    def operar(self):
        self.cliente1.depositar(100)
        self.cliente2.depositar(150)
        self.cliente3.depositar(200)
        self.cliente3.extraer(150)

    def depositos_totales(self):

total=self.cliente1.retornar_monto()+self.cliente2.retornar_monto()+self.clie
nte3.retornar_monto()
        print("El total de dinero del banco es:",total)
        self.cliente1.imprimir()
        self.cliente2.imprimir()
        self.cliente3.imprimir()


# bloque principal

banco1=Banco()
banco1.operar()
banco1.depositos_totales()
```

Recordemos que en Python para diferenciar un atributo de una variable local o un parámetro le antecedemos la palabra clave self (es decir nombre es el parámetro y self.nombre es el atributo): self.nombre=nombre

(la clase Cliente colabora con la clase Banco):

```
class auto:

    def __init__(self, marca,color,ruedas):
        self.marca=marca
#pasamos valores de los parametros a las variables
        self.color=color
        self.ruedas=ruedas

class caracteristicas:
    def __init__(self):
        self.vehiculo= auto("TOYOTA", "AZUL", 4)
        print("Datos del Vehículo")
        print("Marca: ", self.vehiculo.marca,"\nColor: ",self.vehiculo.color, "\nRuedas: ", self.vehiculo.ruedas)
        if self.vehiculo.ruedas==1:
            print("El vehículo es un Monociclo")
        elif self.vehiculo.ruedas==2:
            print("El vehículo es una Moto")
        elif self.vehiculo.ruedas==3:
            print("El vehículo es un Triciclo")
        elif self.vehiculo.ruedas==4:
            print("El vehículo es un Auto")
        elif self.vehiculo.ruedas>4:
            print("El vehículo es un Camión")

objeto=caracteristicas()
```

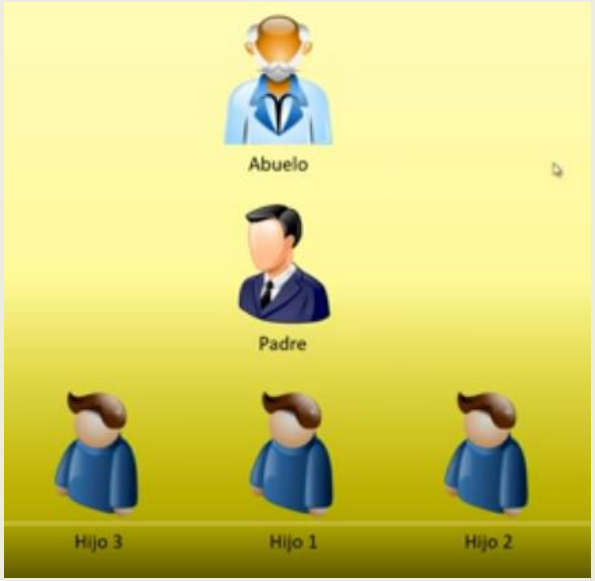
Los atributos tiene self adelante

La clase auto colabora con la clase características.

HERENCIA

La **herencia** es un proceso mediante el cual se puede crear una clase **hija** que hereda de una clase **padre**, compartiendo sus métodos y atributos. Además de ello, una clase hija puede sobreescribir los métodos o atributos, o incluso definir unos nuevos.
Se puede crear una clase hija con tan solo pasar como parámetro la clase de la que queremos heredar.

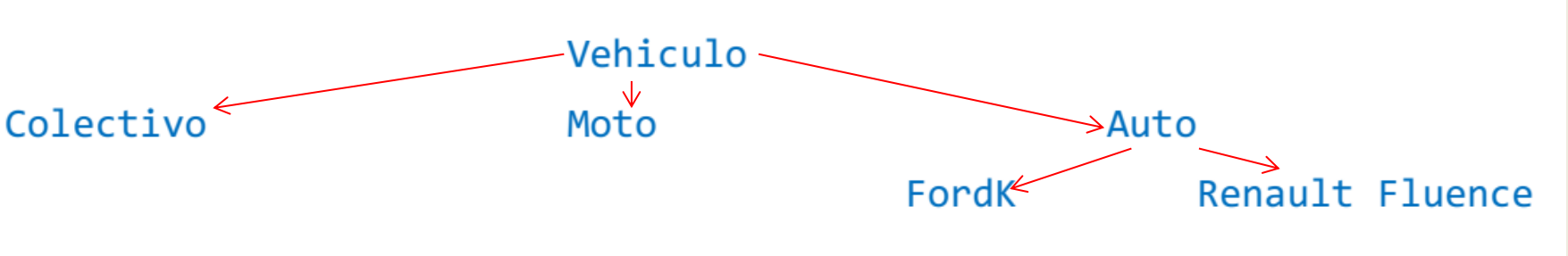
JERARQUIA DE HERENCIA



La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todos los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios

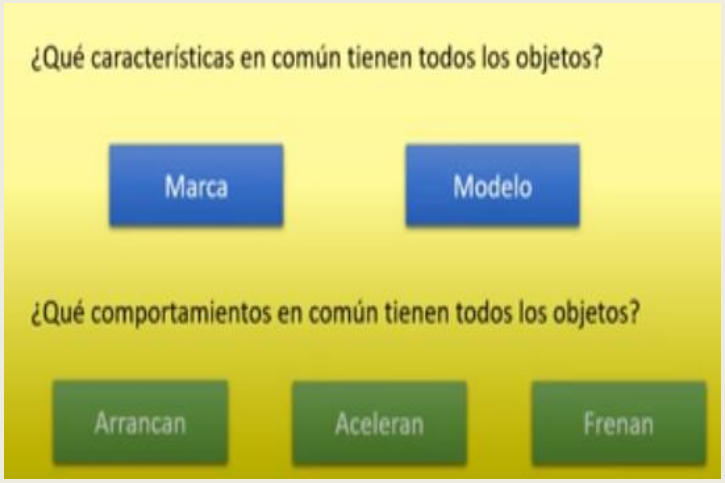
clase padre
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la clase padre.
Subclase
Clase descendiente de otra. Hereda automáticamente los atributos y métodos de su superclase.
Es una especialización de otra clase.
Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

Retomamos el ejemplo de la clase anterior
Imaginemos la clase Vehículo. Qué clases podrían derivar de ella?



¿Para qué sirve la herencia?

Para la reutilización de código en caso de crear objetos similares



Si la herencia no existiera tendría que crear cada objeto de a uno.

ATRIBUTOS

METODOS

HERENCIA

```
class Vehiculos():

    def __init__(self, marca, modelo):

        self.marca=marca
        self.modelo=modelo
        self.enmarcha=False
        self.acelera=False
        self.frena=False

    def arrancar(self):

        self.enmarcha=True

    def acelerar(self):
        self.acelera=True
```

Definimos la Clase Vehículos

```
    def frenar(self):
        self.frena=True

    def estado(self):
        print ("Marca: ", self.marca, "\nModelo: ", self.modelo, "\nEn Marcha: ",
            self.enmarcha, "\nAcelarando: ", self.acelera, "\nFrenando: " , self.frena)
```

```
class Moto(Vehiculos):
    pass
```

La clase moto es heredada de vehículos. Es por esto que al definir la clase entre () ponemos el nombre de la Clase de la cuál heredará (vehículos)

```
miMoto=Moto("Honda", "CBR")
```

Luego creo una instancia de la clase Moto que va a heredar todo lo de la clase vehículo

Está heredando todo inclusive el constructor. Por ello, debemos pasarle los parámetros los valores de marca y motor

```
miMoto.estado()
```

desde la instancia de la clase Moto llamo al método de la Clase vehículo porque es heredada.