TI 3001 C

Analítica de datos y herramientas de inteligencia artificial

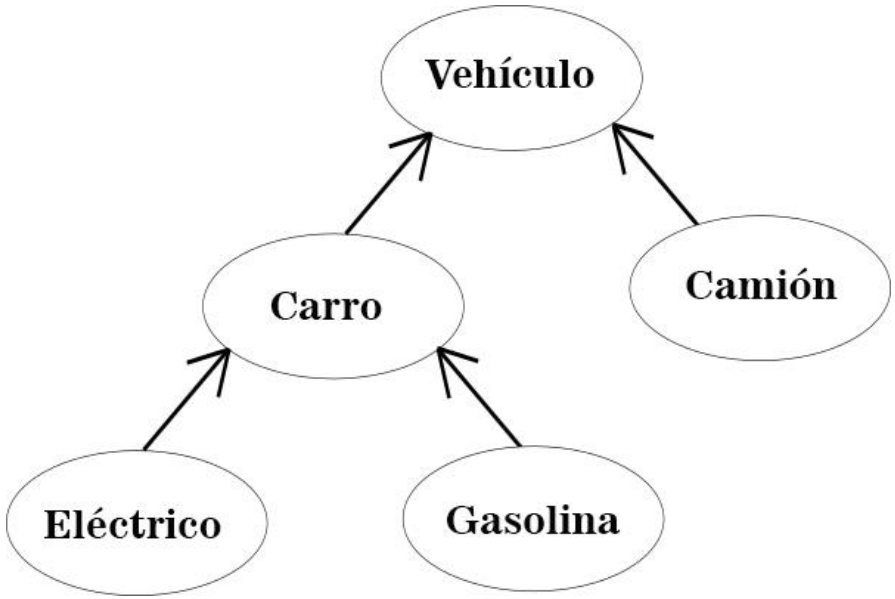
POO: Herencia

Tecnológico de Monterrey

Herencia

La herencia es la capacidad de reutilizar una clase extendiendo su funcionalidad. Una clase que hereda de otra puede añadir nuevos atributos, ocultarlos, añadir nuevos métodos o redefinirlos.

- Término que proviene de la herencia de características particulares como color de ojos, color de pelo, etc.
- Cuando una clase hereda las variables y métodos de otra clase, se dice que la primera es una subclase de la segunda.



Herencia

Una clase que hereda de otra puede redefinir los métodos de la clase padre.

La subclase hereda los métodos de la superclase, a menos que la subclase los reimplemente. Cuando una clase hija hereda de la clase padre un método en particular se puede **reescribir ese método**.

Ejemplo:

- Supongamos, la clase padre tiene el método **mostrarDatos()** y la clase hija también tiene implementado el método **mostrarDatos()**.
- Cuando creamos un objeto de la clase hija y llamamos al método **mostrarDatos()** va a estar implementado con el código que esté dentro de la clase hija y no de la clase padre. Ya que el método se reescribió.

Ejemplo: Herencia

```python
class Empleado:
    def __init__(self, nombre, edad, sueldo):
        self.nombre = nombre
        self.edad = edad
        self.sueldoBase = sueldo
    # Todos los empleados tienen estos atributos

    def calcularSueldo(self, descuentos, bonos):
        return self.sueldoBase - descuentos + bonos
    # Todos los empleados tiene un método para calcular su sueldo.

    def mostrarDatos(self):
        return ("Mi nombre es " + self.nombre + " tengo " + str(self.edad) + " años " +
                " y mi sueldo base es " + str(self.sueldoBase))

def main():
    pedro = Empleado("Pedro Pérez", 25, 25000)
    print(pedro.nombre)
    print(pedro.edad)
    print(pedro.sueldoBase)
    print(pedro.mostrarDatos())
    print("Sueldo neto: ", pedro.calcularSueldo(200, 2000))

main()
```

Shell

```
>>> %Run clase_Empleado0.py
 Pedro Pérez
 25
 25000
 Mi nombre es Pedro Pérez tengo 25 años  y mi sueldo base es 25000
 Sueldo neto:  26800
```

Ejemplo: Herencia

```python
class Empleado:
    def __init__(self, nombre, edad, sueldo):
        self.nombre = nombre
        self.edad = edad
        self.sueldoBase = sueldo
    # Todos los empleados tienen estos atributos

    def calcularSueldo(self, descuentos, bonos):
        return self.sueldoBase - descuentos + bonos
    # Todos los empleados tiene un método para calcular su sueldo.

class AgenteVentas(Empleado): # Para que la clase agente de ventas
    # herede de la clase Empleado, se pone entre paréntesis la clase
    # de la cual hereda.
    # La clase AgenteVentas hereda todos los atributos y métodos de
    # la clase empleado. También hereda el método init, pero si reimplementamos
    # un método, se usa el de la clase hija y no el de la clase padre.
    # Si redefinimos el método init en la clase hija, se toma el método
    # de la clase hija.

    def __init__(self, mostrador):
        self.numeroMostrador = mostrador
        # Este agente de ventas está asignado a un mostrador.

def main():
  pedro = AgenteVentas(8)
  print("El mostrador es:", pedro.numeroMostrador)

main()
```

```
Shell
El mostrador es: 8
```

Los atributos **nombre, edad y sueldo** son parte del constructor de la clase padre **Empleado**, pero el constructor se redefinió y estos atributos no están incluidos.

Así como está definido, la clase **AgenteVentas** solamente utiliza el método **init** de la clase hija.

Ejemplo: Herencia

```python
class Empleado:
    def __init__(self, nombre, edad, sueldo):
        self.nombre = nombre
        self.edad = edad
        self.sueldoBase = sueldo
    # Todos los empleados tienen estos atributos

    def calcularSueldo(self, descuentos, bonos):
        return self.sueldoBase - descuentos + bonos
    # Todos los empleados tiene un método para calcular su sueldo.

class AgenteVentas(Empleado): # Para que la clase agente de ventas
    # herede de la clase Empleado, se pone entre paréntesis la clase
    # de la cual hereda.
    # La clase AgenteVentas hereda todos los atributos y métodos de
    # la clase empleado. También hereda el método init, pero si reimplementamos
    # un método, se usa el de la clase hija y no el de la clase padre.
    # Si redefinimos el método init en la clase hija, se toma el método
    # de la clase hija.

    def __init__(self, mostrador):
        self.numeroMostrador = mostrador
        # Este agente de ventas está asignado a un mostrador.

def main():
    pedro = AgenteVentas(4)
    # Se instancia un objeto de tipo AgenteVenas, el constructor recibe el
    # número de mostrador
    print(pedro.nombre)
main()
```

```
Shell
Traceback (most recent call last):
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\clase
    main()
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\clase
    print(pedro.nombre)
AttributeError: 'AgenteVentas' object has no attribute 'nombre'
```

Suponemos que **pedro** tiene un atributo **nombre**. Pero no estamos en lo correcto, marca un error, nos dice que la clase **AgenteVentas** no tiene el atributo **nombre**. Esto es así porque se redefinió el método **init** donde solamente se tiene acceso a los atributos de la clase hija (**mostrador**).

El atributo **nombre** es parte del constructor de la clase padre **Empleado**, pero el constructor se redefinió y este atributo no está incluido.

Ejemplo: Herencia

```python
class Empleado:
    def __init__(self, nombre, edad, sueldo):
        self.nombre = nombre
        self.edad = edad
        self.sueldoBase = sueldo
    # Todos los empleados tienen estos atributos

    def calcularSueldo(self, descuentos, bonos):
        return self.sueldoBase - descuentos + bonos
    # Todos los empleados tiene un método para calcular su sueldo.

class AgenteVentas(Empleado): # Para que la clase agente de ventas
    # herede de la clase Empleado, se pone entre paréntesis la clase
    # de la cual hereda.
    # La clase AgenteVentas hereda todos los atributos y métodos de
    # la clase empleado. También hereda el método init, pero si reimplementamos
    # un método, se usa el de la clase hija y no el de la clase padre.
    # Si redefinimos el método init en la clase hija, se toma el método
    # de la clase hija.
    def __init__(self, nombre, edad, sueldo, mostrador):
        self.numeroMostrador = mostrador
        super().__init__(nombre, edad, sueldo)
        # Se llama con la palabra reservada super().__init__()
        # Este agente de ventas está asignado a un mostrador.

def main():
    pedro = AgenteVentas("Pedro López", 25, 20000, 4)
    print(pedro.nombre)

main()
```

```
Shell
>>> %Run clase_Empleado1_2.py
Pedro López
```

- Debemos llamar el método **init** de la clase padre (super clase).
- Se llama a la clase padre con la palabra **super().__init__()**. El método **init** recibe tres parámetros de la clase padre, que se copian al constructor de la clase hija y pasan como parámetro en la llamada el método **init** de la clase padre.
- Se le envían los parámetros al constructor de la superclase.

Ejemplo: Herencia

```python
class Empleado:
    def __init__(self, nombre, edad, sueldo):
        self.nombre = nombre
        self.edad = edad
        self.sueldoBase = sueldo

    def calcularSueldo(self, descuentos, bonos):
        return self.sueldoBase - descuentos + bonos

    def mostrarDatos(self):
        return ("Mi nombre es " + self.nombre + " tengo " + str(self.edad) +
                " años " + "y mi sueldo base es " + str(self.sueldoBase))

class AgenteVentas(Empleado):

    def __init__(self, nombre, edad, sueldo, mostrador):
        self.numeroMostrador = mostrador
        super().__init__(nombre, edad, sueldo)
        # Se llama con la palabra reservada super().__init__()
        # Este agente de ventas está asignado a un mostrador.

    def mostrarDatos(self):
        return ("Mi nombre es " + self.nombre + ", tengo " + str(self.edad) +
                " años," + "mi sueldo es " + str(self.sueldoBase) +
                "y estoy en el mostrador " + str(self.numeroMostrador))
def main():
    pedro = AgenteVentas("Pedro López", 25, 20000, 4)
    print(pedro.nombre)
    print("Sueldo neto:", pedro.calcularSueldo(200, 2000))
    print(pedro.mostrarDatos())
main()
```

Shell

```
>>> %Run clase_Empleado1_4.py
Pedro López
Sueldo neto: 21800
Mi nombre es Pedro López, tengo 25 años,mi sueldo es 20000y estoy en el mostrador 4
```

La clase **AgenteVentas** puede redefinir el método **mostrarDatos**.

Ejemplo: Herencia

```python
class Empleado:
    def __init__(self, nombre, edad, sueldo):
        self.nombre = nombre
        self.edad = edad
        self.sueldoBase = sueldo

    def calcularSueldo(self, descuentos, bonos):
        return self.sueldoBase - descuentos + bonos

    def mostrarDatos(self):
        return ("Mi nombre es " + self.nombre + " tengo " + str(self.edad) +
                " años " + "y mi sueldo base es " + str(self.sueldoBase))

class Tripulante(Empleado): # hereda de Empleado
    # No va a tener ningún atributo específico, solamente va
    # heredar los atributos de la clase padre.
    # Por lo que no implementaremos un construtor.
    # Dejamos que lo tome de la clase empleado.

    def mostrarRenovacionLicencia(self): # No recibe nada
        if self.edad < 50:
            print("Renueva su licencia cada año")
        else:
            print("Renueva su licencia cada 6 meses")

def main():
    luis = Tripulante("Luis Garza", 45, 40000)
    # Instancio objeto de tipo tripulante
    luis.mostrarRenovacionLicencia()
    print(luis.mostrarDatos())
    print("Sueldo neto:", luis.calcularSueldo(3000, 150))
main()
```

```
Shell
>>> %Run clase_Empleado1_5.py
Renueva su licencia cada año
Mi nombre es Luis Garza tengo 45 años y mi sueldo base es 40000
Sueldo neto: 37150
```

Agregamos la clase **Tripulante**. Hereda de **Empleado**. Hereda los atributos de la clase padre. **No se redefine el constructor.**

Se instancia un objeto de tipo **Tripulante**. Se pasan todos los argumentos que requiere el constructor de la clase padre.

El método **mostrarRenovacionLicencia** muestra cada cuanto se tiene que renovar la licencia.

Herencia múltiple

Una clase puede heredar de más de una clase a la vez.

```
class A:
    def print_a(self):
        print('a')


class B:
    def print_b(self):
        print('b')


class C(A, B):
    def print_c(self):
        print('c')


c = C()
c.print_a()
c.print_b()
c.print_c()
```

El script dará como resultado:

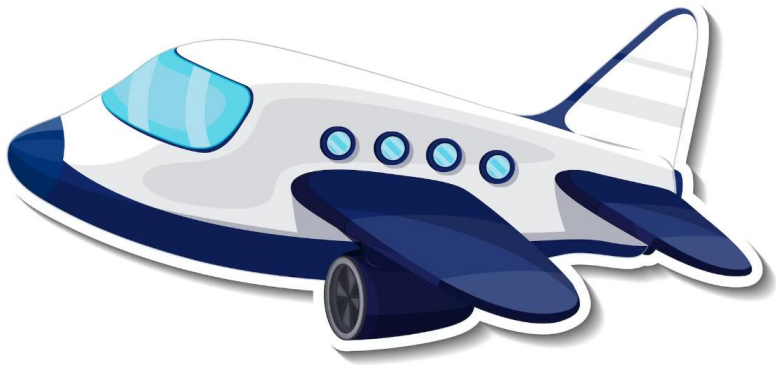
```
1.  a
2.  b
3.  c
```

Ejercicio: Herencia

- Define la clase **Coche**.
- Define el atributo de clase **ruedas** con el valor de 4.
- Define en el **constructor** los atributos de instancia **color**, **aceleración** y **velocidad**. Los atributos **color** y **aceleración** reciben su valor con los parámetros de entrada. La velocidad se inicializa en 0 dentro del constructor.
- Crea el método **acelera** que modifique la velocidad (velocidad + aceleración).
- Crea el método **frena** que modifique la velocidad (velocidad - aceleración). Si la velocidad es menor a 0, regresar la velocidad 0.
- Crea el método **mostrar** que regrese en un string la concatenación de los atributos de instancia (color, aceleración y velocidad) y el atributo de clase (ruedas).

Ejercicio: Herencia

- Define la clase **CocheVolador** como una clase hija de la clase **Coche**.
- Define el atributo de clase **ruedas** con el valor de 6.
- Define el método **constructor** con los atributos de instancia de la super clase **Coche** y el atributo de instancia de la clase **CocheVolador**: **esta_volando = False**.
- Crea el método **vuela** que modifique el atributo **esta_volando** a **True**.
- Crea el método **aterriza** que modifique el atributo **esta_volando** a **False**.
- Crea el método **mostrar** que regrese en un string la concatenación de los atributos de instancia y clase.



Herencia

La herencia es la capacidad de reutilizar una clase extendiendo su funcionalidad. Una clase que hereda de otra puede añadir nuevos atributos, ocultarlos, añadir nuevos métodos o redefinirlos.

Ejemplo: La clase **CocheVolador** hereda de la clase **Coche**.

```
class Coche:

ruedas = 4

def __init__(self, color, aceleracion):
    self.color = color
    self.aceleracion = aceleracion
    self.velocidad = 0

def acelera(self):
    self.velocidad = self.velocidad + self.aceleracion

def frena(self):
    v = self.velocidad - self.aceleracion
    if v < 0:
        v = 0
    self.velocidad = v
```

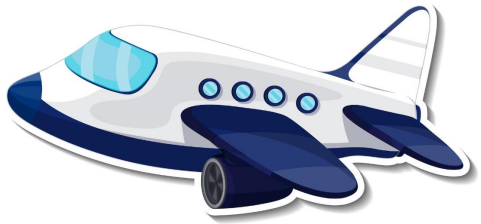
```
class CocheVolador(Coche):

ruedas = 6

def __init__(self, color, aceleracion, esta_volando = False):
    super().__init__(color, acelaracion)
    self.esta_volando = esta_volando

def vuela(self):
    self.esta_volando = True

def aterriza(self):
    self.esta_volando =  False
```



Herencia

Ejemplo de herencia: La clase **CocheVolador** hereda de la clase **Coche**. El nombre de la clase padre se indica entre paréntesis a continuación del nombre de la clase hija.

```
class Coche:

    ruedas = 4

    def __init__(self, color, aceleracion):
        self.color = color
        self.aceleracion = aceleracion
        self.velocidad = 0

    def acelera(self):
        self.velocidad = self.velocidad + self.aceleracion

    def frena(self):
        v = self.velocidad - self.aceleracion
        if v < 0:
            v = 0
        self.velocidad = v
```

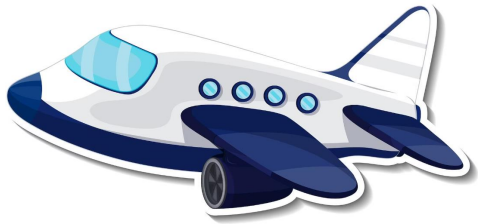
```
class CocheVolador(Coche):

    ruedas = 6

    def __init__(self, color, aceleracion, esta_volando = False):
        super().__init__(color, acelaracion)
        self.esta_volando = esta_volando

    def vuela(self):
        self.esta_volando = True

    def aterriza(self):
        self.esta_volando =  False
```



Herencia

La clase **CocheVolador** redefine el atributo de clase **ruedas**, estableciendo su valor a 6 e implementa dos métodos nuevos: **vuela()** y **aterriza().**

```
class Coche:

ruedas = 4

def __init__(self, color, aceleracion):
    self.color = color
    self.aceleracion = aceleracion
    self.velocidad = 0

def acelera(self):
    self.velocidad = self.velocidad + self.aceleracion

def frena(self):
    v = self.velocidad - self.aceleracion
    if v < 0:
        v = 0
    self.velocidad = v
```

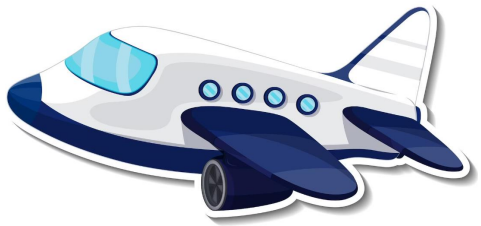
```
class CocheVolador(Coche):

ruedas = 6

def __init__(self, color, aceleracion, esta_volando = False):
    super().__init__(color, acelaracion)
    self.esta_volando = esta_volando

def vuela(self):
    self.esta_volando = True

def aterriza(self):
    self.esta_volando = False
```



Herencia

En la primera línea del método **__init__()** aparece la función **super()**. Esta función devuelve un objeto temporal de la superclase (**Coche**) que permite invocar a los métodos definidos en la misma. Se redefine el método **__init__()** de la clase hija usando la funcionalidad del método de la clase padre. Como la clase **Coche** es la que define los atributos **color** y **aceleracion**, estos se pasan al constructor de la clase padre y, a continuación, se crea el atributo de instancia **esta_volando** solo para objetos de la clase **CocheVolador.**

```
class Coche:                                   ← Clase padre

ruedas = 4

def __init__(self, color, aceleracion):
    self.color = color
    self.aceleracion = aceleracion
    self.velocidad = 0

def acelera(self):
    self.velocidad = self.velocidad + self.aceleracion

def frena(self):
    v = self.velocidad - self.aceleracion
    if v < 0:
        v = 0
    self.velocidad = v
```

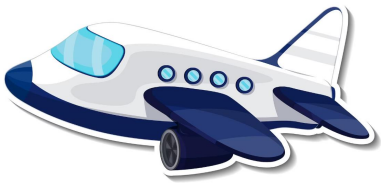
```
class CocheVolador(Coche):                     ← Clase hija

ruedas = 6

def __init__(self, color, aceleracion, esta_volando = False):
    super().__init__(color, acelaracion)
    self.esta_volando = esta_volando

def vuela(self):
    self.esta_volando = True

def aterriza(self):
    self.esta_volando =  False
```

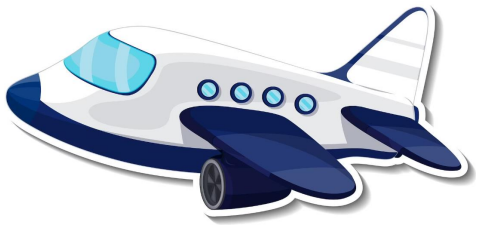


Herencia

Al utilizar la herencia, todos **los atributos (atributos de datos y métodos)** de la **clase padre** también pueden ser referenciados por objetos de las **clases hijas**. Al revés no ocurre lo mismo.

```
>>> c = Coche('azul', 10)
>>> cv1 = CocheVolador('rojo', 60)
>>> print(cv1.color)
rojo
>>> print(cv1.esta_volando)
False
>>> cv1.acelera()
>>> print(cv1.velocidad)
60
>>> print(CocheVolador.ruedas)
6
>>> print(c.esta_volando)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AttributeError: 'Coche' object has no attribute 'esta_volando'
```

```
>>> cv1.vuela()
>>> print(cv1.esta_volando)
True
```



```
class Coche:

ruedas = 4

def __init__(self, color, aceleracion):
    self.color = color
    self.aceleracion = aceleracion
    self.velocidad = 0

def acelera(self):
    self.velocidad = self.velocidad + self.aceleracion


class CocheVolador(Coche):

ruedas = 6

def __init__(self, color, aceleracion, esta_volando = False):
    super().__init__(color, acelaracion)
    self.esta_volando = esta_volando

def vuela(self):
    self.esta_volando = True

def aterriza(self):
    self.esta_volando =  False
```

Gracias