ICAI
Instituto de Capacitación y Asesoría en Informática

Instituto de Capacitación y Asesoría en Informática de la Escuela de Informática

UNA UNIVERSIDAD NACIONAL COSTA RICA

ICAI
Instituto de Capacitación y Asesoría en Informática

Programación en Python Básico

Ing Luis Diego Gamboa Chaverri

UNA UNIVERSIDAD NACIONAL COSTA RICA

Agenda del día

- Conceptos de Orientación de Objetos
- Clase e instancias
- Creación de Objetos
- Métodos de Clase
- Encapsulamiento
- Ocultamiento
- Herencia de Clases
- Polimorfismo

¿Por qué es tan importante la programación orientada a objetos?

Nuevamente lo sé!, pero recuerde que en Python todo es un objeto

Este tipo de programación introduce un nuevo paradigma que permite encapsular , aislar datos y operaciones que se pueden realizar sobre dichos datos u objetos

Objetos

- Si miramos a nuestro alrededor , se pueden observar muchos objetos : una silla, la cocina, una bicicleta, nuestra mascota, etc....
- Cada uno de estos objetos poseen características como **Estado** y **Comportamiento**

Estado: llantas, número de marchas, marcha actual, asientos, etc.
Comportamiento: acelerar, frenar, cambiar de marcha, etc.

Estado: nombre, color, raza, etc.
Comportamiento: galopar, relinchar, comer, beber, etc.

Objetos

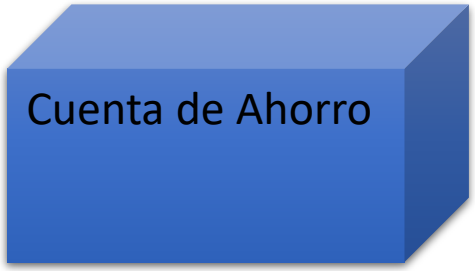
En software los objetos son modelos de los objetos del mundo real, por lo cual también poseen estados y comportamientos

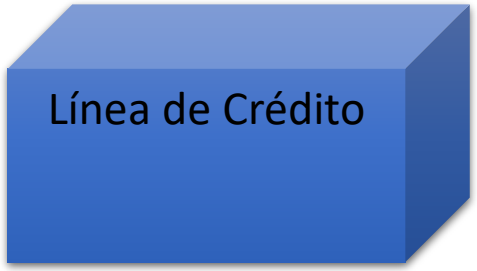
Los **Estados** son mantenidos en **variables** y los **Comportamientos** son implementados en **métodos**

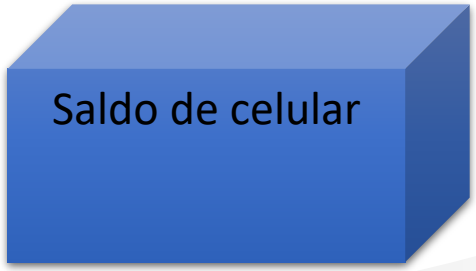
Objetos

Representación de Objetos Real

Objetos abstractos



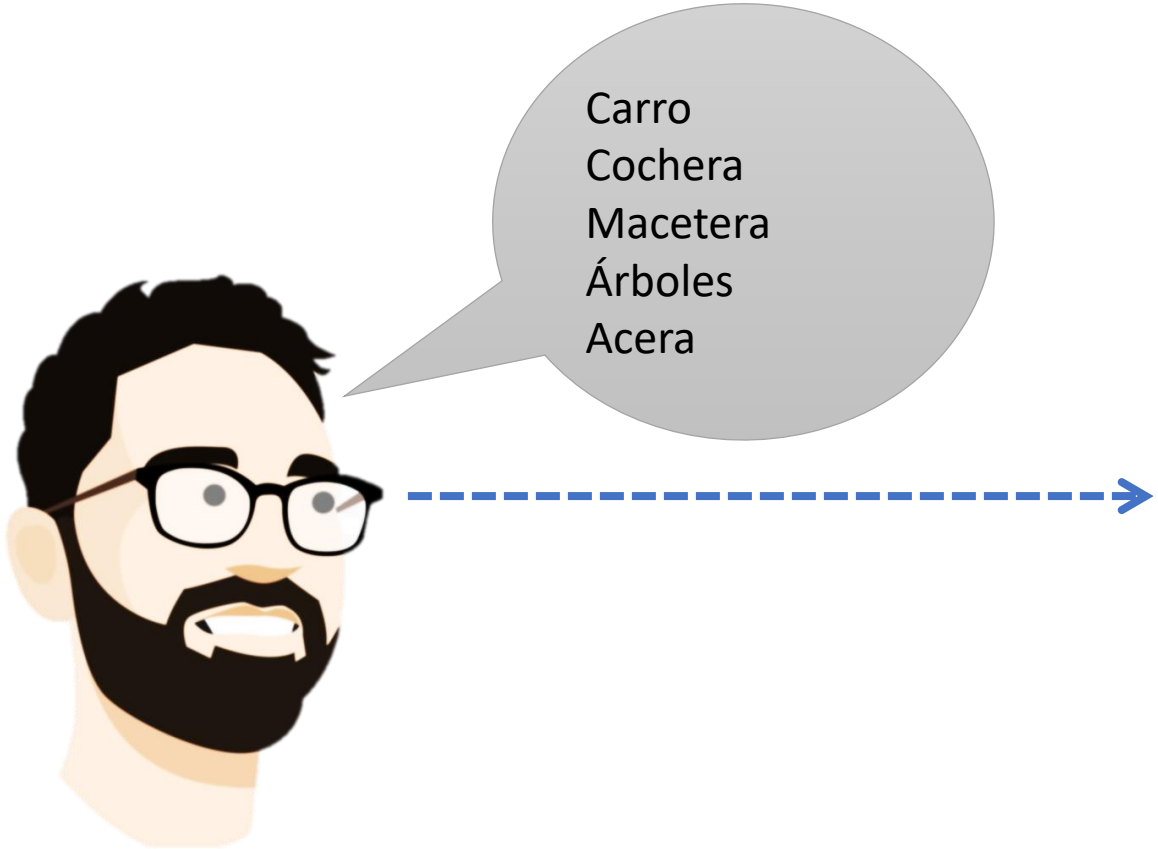




Objetos

Identificar Objetos puede verse como un arte y no una ciencia, y el resultado dependerá del contexto y punto de vista de quien modele

Qué deseamos Modelar?



Interactuar con el Objeto

Emisor

1. Se acerca al auto y oprime el control remote para abrirlo
2. El auto se abre y la persona entra
3. La persona Cierra la Puerta
4. La persona inserta la llave
5. La persona enciende el auto
6. La persona cambia de marcha
7. La persona acelera
8. El auto avanza

Receptor

Interactuar con el Objeto

La persona se comunicó con el auto

Emisor

- A través de un **mensaje** abrio el auto
- El auto tuvo la **responsabilidad** de abrirse
- La persona sube al auto y por medio de un **mensaje** enciende el auto
- El auto tiene la **responsabilidad** de encenderse
- La persona envia un **mensaje** para cambiar de marcha
- El auto tiene la **responsabilidad** de cambiar de marcha
- La persona envia el **mensaje** de acelerar
- El auto tiene la **responsabilidad** de acelerar

Receptor

Estados y comportamientos

Estado

- Apagado / Encedido
- Puestas abiertas / cerradas
- Cantidad de ocupantes
- Marcha actual
- Manual / automático
- Min y Max Velocidad

Comportamiento

- Abrir / Cerrar puertas
- Encender / Apagar auto
- Cambiar marcha
- Acelerar
- Frenar

Comó se comunican los estados con los comportamientos?

Interactuar con el Objeto

Emisor

- Abrir auto
- Abrir Puerta
- Cerrar Puerta
- Encender auto
- Cambiar marcha
- Acelerar

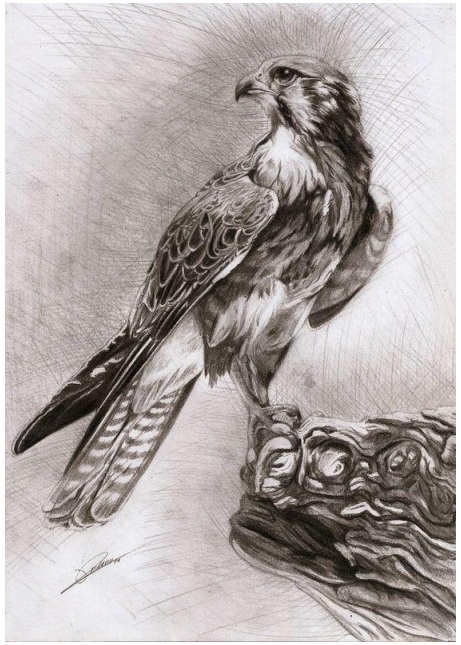
Receptor

La interpretación del mensaje (método usando para responder al mensaje) es determinado por el receptor

Interactuar con el Objeto

Emisor

- Abrir auto
- Abrir Puerta
- Cerrar Puerta
- Encender auto
- Cambiar marcha
- acelerar



Receptor

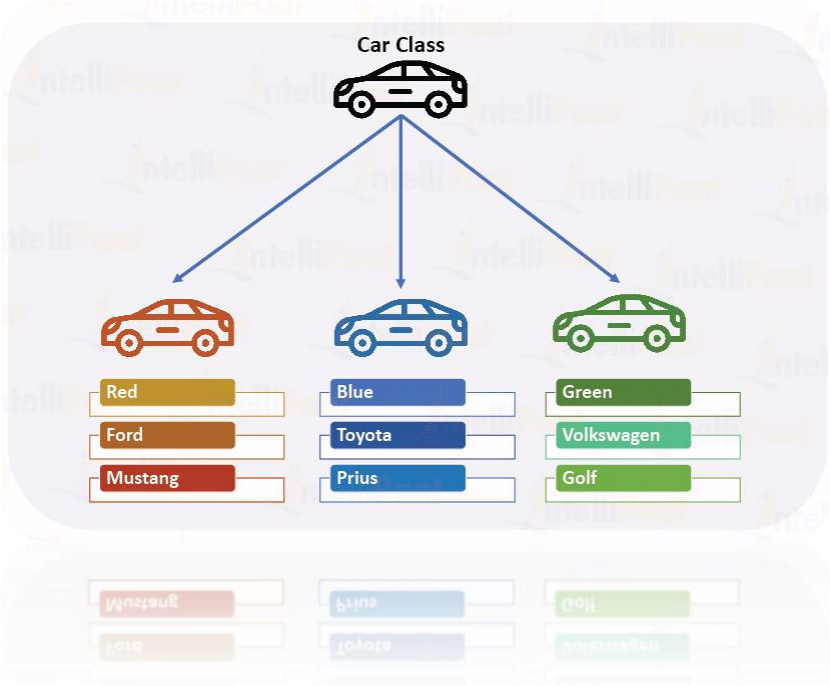
ICAI
Instituto de Capacitación y Asesoría en Informática

POO en Python

UNA
UNIVERSIDAD NACIONAL
COSTA RICA

POO en Python permite entre otros

- Definir nuevas clases
- Instanciar clases
- Heredar entre clases



Clases e instancias

Una **clase** es un molde a partir del cual se crean **instancias** con las mismas características y comportamiento

Por ejemplo la persona y otras personas que usan el auto son **Instancias** de la clase y tienen características comunes como saber abrir encender el auto, acelerar , etc.

Por ende se puede decir que todos ellos pertenecen a una categoría o **clase** que se puede denominar “conductor”

Clases

Una clase es una entidad que define una serie de elementos que determinan un estado (datos) y un comportamiento (operaciones sobre los datos que modifican su estado).

```
class Conductor:
    def __init__(self,nombre,licencia):
        __edad = 18
        __categoria = "novato"
        __puntos = 10

        self.__nombre = nombre
        self.__licencia = licencia
        self.__edad = __edad
        self.disponible = True
        self.__categoria = __categoria
        self.__puntos = __puntos

    def getNombre(self):
        return self.__nombre
    def getLicencia(self):
        return self.__licencia
    def setNombre(self,nombre):
        self.__nombre = nombre
    def setLicencia(self,licencia):
        self.__licencia = licencia
```

Clases en Python

- Inicia con la palabra reservada **class,** seguida del **nombre** y finalizando con **:**
- Los nombres deben comenzar con **mayúscula**
- Dentro de ella todo debe ir con **sangría**

```
class Conductor:
    def __init__(self,nombre,licencia):
        __edad = 18
        __categoria = "novato"
        __puntos = 10

        self.__nombre = nombre
        self.__licencia = licencia
        self.__edad = __edad
        self.disponible = True
        self.__categoria = __categoria
        self.__puntos = __puntos

    def getNombre(self):
        return self.__nombre
    def getLicencia(self):
        return self.__licencia
    def setNombre(self,nombre):
        self.__nombre = nombre
    def setLicencia(self,licencia):
        self.__licencia = licencia
```

Instancias en Python

- Se utiliza la misma notación de funciones
- Nombre + Paréntesis de apertura + Argumentos + Paréntesis de cierre

```
ana   = Conductor('Ana Maria',888888)
luis = Conductor('Luis Eduardo',7777777)
juan = Conductor("Juan Felipe",9999999)
```

Contructores

- Son métodos especiales que se usan para **inicializar** la instancia
- Estos se ejecutan justo después de **instanciar** un objeto
- Estos se crean mediante el método especial **__init__**
- En este método se inicializan las variables para su posterior uso en los métodos

```
def __init__(self,nombre,licencia):
    __edad = 18
    __categoria = "novato"
    __puntos = 10

    self.__nombre = nombre
    self.__licencia = licencia
    self.__edad = __edad
    self.disponible = True
    self.__categoria = __categoria
    self.__puntos = __puntos
```

Contructores

```
def __init__(self,nombre,licencia):
    __edad = 18
    __categoria = "novato"
    __puntos = 10

    self.__nombre = nombre
    self.__licencia = licencia
    self.__edad = __edad
    self.disponible = True
    self.__categoria = __categoria
    self.__puntos = __puntos
```

```
ana  = Conductor('Ana Maria',888888)
luis = Conductor('Luis Eduardo',7777777)
juan = Conductor("Juan Felipe",9999999)
```

Self

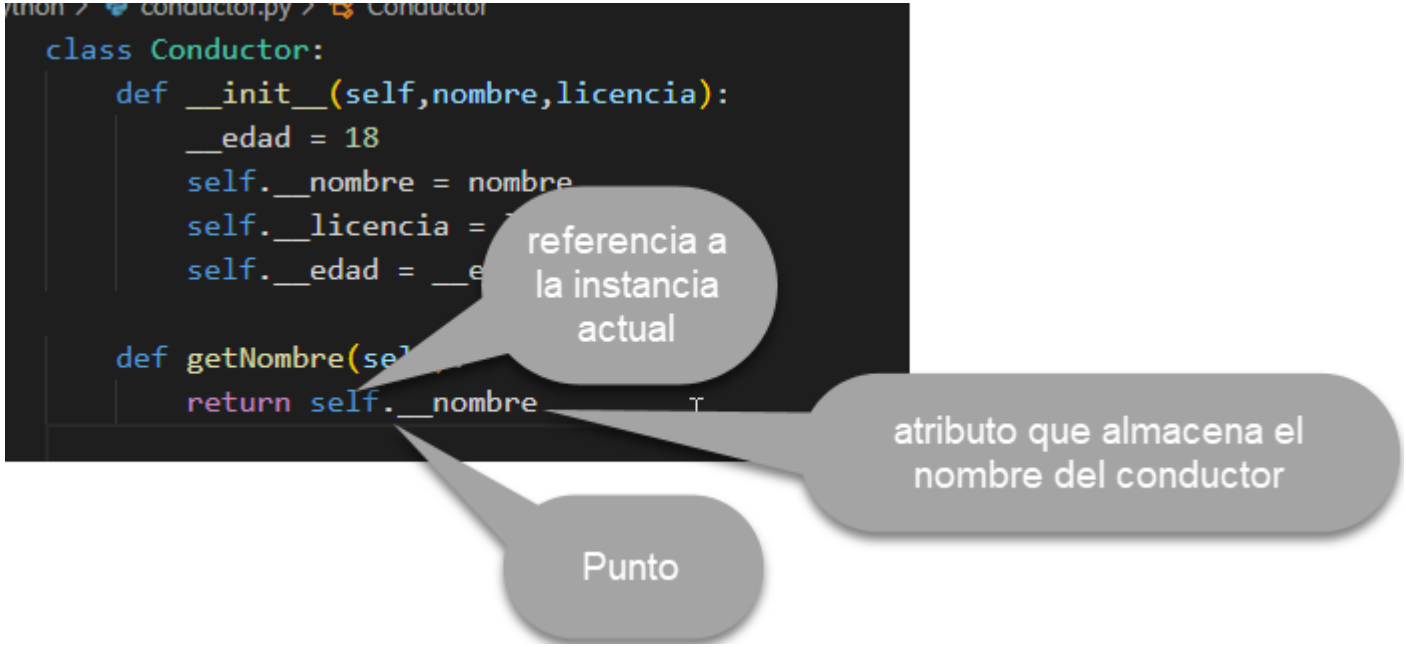
- La palabra reservada **self** hace referencia al objeto actual, y se utiliza dentro del propio objeto para señalarse a si mismo
- En todos los métodos de una clase, el primer parámetro es siempre **self**
- Es necesaria para poder tener acceso a los atributos y métodos del objeto, evitando la ambigüedad.

```
def __init__(self,nombre,licencia):
    __edad = 18
    self.__nombre = nombre
    self.__licencia = licencia
    self.__edad = __edad
```

Elimina ambiguedad entre Variable de instancia o variable local

Atributos

- También llamados **variables de instancia**
- Son los **contenedores** del estado del objeto
- Son almacenes de datos
- Para almacenar en ellos se usa la notación de **punto**



Métodos

- Similares a las funciones, sólo que su definición se encuentra en la clase
- El primer parámetro es siempre **self**

```
class Conductor:
    def __init__(self,nombre,licencia):
        __edad = 18
        __categoria = "novato"
        __puntos = 10

        self.__nombre = nombre
        self.__licencia = licencia
        self.__edad = __edad
        self.disponible = True
        self.__categoria = __categoria
        self.__puntos = __puntos

    def getNombre(self):
        return self.__nombre
    def getLicencia(self):
        return self.__licencia
    def setNombre(self,nombre):
        self.__nombre = nombre
    def setLicencia(self,licencia):
        self.__licencia = licencia

    def toString(self):
        return "Nombre: " + self.__nombre + "Licencia: " + self.__licencia
    def __cambiar_categoria(self, categoria):
        self.__categoria = categoria
    def subir_puntos(self):
        self.__puntos = self.__puntos - 1
    def bajar_puntos(self):
        self.__puntos = self.__puntos + 1
```

Invocación de Métodos de la clase

- Se utiliza la notación de punto
- Se omite el parámetro self, ya que es implícito

```
juan = Conductor("Juan Felipe",9999999)

juan.subir_puntos()
```

Encapsulamiento

- Es como construir una “cápsula” que hace las de barrera entorno a la información y las operaciones de los objetos
- Permite que las cosas permanezcan juntas
- Es la que permite que limitamos el acceso o damos un acceso restringido de una propiedad a los elementos que necesita un miembro y no a ninguno más.
- **Ejemplo** : Clases , los getters y setters

Ocultamiento

- Se refiere a la posibilidad de ocultar detalles del comportamiento de un objeto.
- O viéndolo de otra manera, mostrar solamente lo que nos interesa
- Esto se conoce como poner público o privado un elemento
- En Python para poner en **privado** un atributo o método, el nombre de este inicia con **doble guión bajo "__" y el nombre del mismo.**

```
class Conductor:
    def __init__(self,nombre,licencia):
        __edad = 18
        __categoria = "novato"
        __puntos = 10

        self.__nombre = nombre
        self.__licencia = licencia
        self.__edad = __edad
        self.disponible = True
        self.__categoria = __categoria
        self.__puntos = __puntos

    def getNombre(self):
        return self.__nombre
    def getLicencia(self):
        return self.__licencia
    def setNombre(self,nombre):
        self.__nombre = nombre
    def setLicencia(self,licencia):
        self.__licencia = licencia

    def toString(self):
        return "Nombre: " + self.__nombre + "Licencia: " + self.__licencia
    def __cambiar_categoria(self, categoria):
        self.__categoria = categoria
    def subir_puntos(self):
        self.__puntos = self.__puntos - 1
    def bajar_puntos(self):
        self.__puntos = self.__puntos + 1
```

Privados

Públicos

```
ana  = Conductor('Ana Maria',888888)
luis = Conductor('Luis Eduardo',7777777)
juan = Conductor("Juan Felipe",9999999)

juan.subir_puntos()
juan.disponible = False
```

Válido

```
juan.__nombre = 10
juan.__cambiar_categoria = 'experto'
```

Inválido

Herencia

- La herencia simple se apoya en el uso de una clase base para compartir sus atributos y comportamientos con otras clases derivadas.

- Una clase que hereda de otra puede añadir nuevos atributos, ocultarlos, añadir nuevos métodos o redefinirlos

```
class Tractorista(Conductor):
    ## constructor
    def __init__(self,nombre,licencia,licencia2):
        ## invoca al constructor de clase Conductor
        super().__init__(nombre,licencia)
        __edad = 21  ## redefine el valor de edad
        ## incluye nuevos atributos
        __experiencia = "novato"

        self.__licencia2 = licencia2
        self.__experiencia = __experiencia


    def getExperiencia(self):
        return self.__experiencia
    def getLicencia2(self):
        return self.__licencia2
    def setExperiencia(self,experiencia):
        self.__experiencia = experiencia
    def setLicencia2(self,licencia):
        self.__licencia2 = licencia
```

Herencia

- La clase **Tractorista** hereda de la clase **Conductor**. En Python, el nombre de la clase padre se indica entre paréntesis a continuación del nombre de la clase hija.

- La clase **Tractorista** redefine atributos como edad , estableciendo su valor en 21 e implementa 4 métodos nuevos: **getExperiencia(), setExperiencia(), setLicencia2(), getLicencia2()**

- La primera línea del método **__init__().** En ella aparece la función **super().** Esta función devuelve un objeto temporal de la superclase (o padre) que permite invocar a los métodos definidos en la misma. Lo que está ocurriendo es que se está redefiniendo el método __init__() de la clase hija usando la funcionalidad del método de la clase padre.
- Como la clase **Conductor** es la que define los atributos **nombre** y **licencia**, estos se pasan al constructor de la clase padre y, a continuación, se crea el atributo de instancia **licencia2** solo para objetos de la clase **Tractorista**.

- 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, pero no al revés.

```
class Tractorista(Conductor):
    ## constructor
    def __init__(self,nombre,licencia,licencia2):
        ## invoca al constructor de clase Conductor
        super().__init__(nombre,licencia)
        __edad = 21  ## redefine el valor de edad
        ## incluye nuevos atributos
        __experiencia = "novato"

        self.__licencia2 = licencia2
        self.__experiencia = __experiencia


    def getExperiencia(self):
        return self.__experiencia
    def getLicencia2(self):
        return self.__licencia2
    def setExperiencia(self,experiencia):
        self.__experiencia = experiencia
    def setLicencia2(self,licencia):
        self.__licencia2 = licencia
```

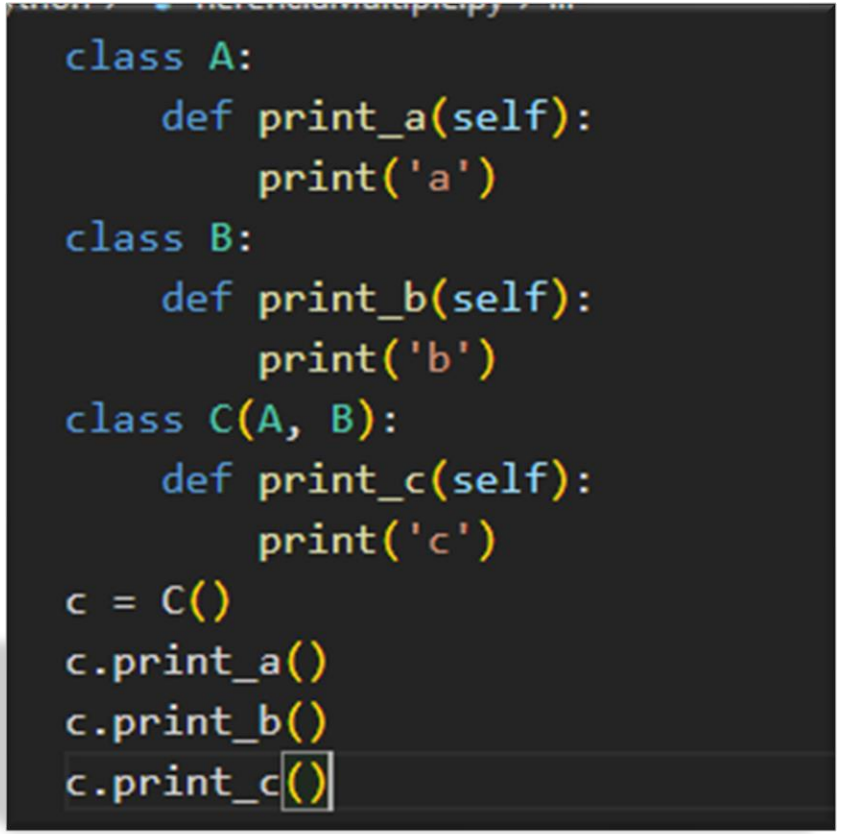
Invocaciones dada la Herencia

```
ana  = Tractorista('Ana Maria',888888,'Trac8888')

ana.subir_puntos()
ana.getLicencia()
ana.getLicencia2()
ana.setLicencia(777777)
print(ana.getLicencia())
```

Herencia Múltiple

La herencia múltiple en Python se dan cuando una clase secundaria o hija hereda atributos y métodos de mas de una clase principal o padre. Basta con separar con una coma ambas principales a la hora de crear la clase secundaria



```
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()
```

Polimorfismo

- *Polimorfismo* es la capacidad de una entidad de referenciar en tiempo de ejecución a instancias de diferentes clases.

```python
class Perro:
    def sonido(self):
        print('Guauuuuu!!!')
class Gato:
    def sonido(self):
        print('Miaaauuuu!!!')

class Vaca:
    def sonido(self):
        print('Múuuuuuuu!!!')


def a_cantar(animales):
    for animal in animales:
        animal.sonido()
if __name__ == '__main__':
    perro = Perro()
    gato = Gato()
    gato_2 = Gato()
    vaca = Vaca()
    perro_2 = Perro()
    granja = [perro, gato, vaca, gato_2, perro_2]
    a_cantar(granja)
```

Polimorfismo

- En el ejemplo se definió una función llamada **a_cantar().** La variable animal que se crea dentro del **ciclo for** de la función es **polimórfica**, ya que en tiempo de ejecución hará referencia a objetos de las clases Perro, Gato y Vaca. Cuando se invoque al **método sonido(),** se llamará al método correspondiente de la clase a la que pertenezca cada uno de los animales.

```
class Perro:
    def sonido(self):
        print('Guauuuuu!!!')
class Gato:
    def sonido(self):
        print('Miaaauuuu!!!')

class Vaca:
    def sonido(self):
        print('Múuuuuuuu!!!')


def a_cantar(animales):
    for animal in animales:
        animal.sonido()
if __name__ == '__main__':
    perro = Perro()
    gato = Gato()
    gato_2 = Gato()
    vaca = Vaca()
    perro_2 = Perro()
    granja = [perro, gato, vaca, gato_2, perro_2]
    a_cantar(granja)
```

ICAI
Instituto de Capacitación y Asesoría en Informática

Instituto de Capacitación y Asesoría en Informática de la Escuela de Informática

icai@una.cr

www.icai.ac.cr

UNA UNIVERSIDAD NACIONAL COSTA RICA