Algorítmica y Programación

Enero - Mayo 2020

Dr. Iván S. Razo Zapata
(ivan.razo@itam.mx)

ITAM

Programación Orientada a Objetos

Temas para esta sesión

- Constructores
- **Encapsulamiento**
- **Herencia**
- **Polimorfismo**

ITAM

Constructores

- Idea general
    - Los objetos creados a veces requieren que sus atributos sean predefinidos antes de poder ser utilizados
- Constructores con y sin paso de parámetros

Constructores

- Ejemplo: Constructor sin parámetros

```
class vehiculo:
    def __init__(self):
        print('Nuevo vehiculo creado.')
```

```
myAuto = vehiculo()
```

Constructores

- Ejemplo: Constructor con parámetros

```
class vehiculo:
    def __init__(self,val1,val2):
        self.modelo = val1
        self.serie  = val2
        print('Nuevo vehiculo creado')
        print('Modelo: ',self.modelo)
        print('Marca: ',self.serie)



myAuto = vehiculo(2020,"X25FG")
```

Constructores

- Ejemplo: Constructor con parámetros y extra atributos de objeto

```
class vehiculo:
    def __init__(self,val1,val2):
        self.modelo = val1
        self.serie  = val2
        self.marca  = "Ford"
        self.color  = "Rojo"
        print('Nuevo vehiculo creado')
        print('Modelo: ',self.modelo)
        print('Marca: ',self.serie)
```

```
myAuto = vehiculo(2020,"X25FG")
```

```
In [8]: myAuto.color
Out[8]: 'Rojo'

In [9]: myAuto.marca
Out[9]: 'Ford'
```

Constructores

- Ejemplo: Constructor con parámetros y extra atributos de clase

```
class vehiculo:
    fabrica = "Puebla"
    pais = "Mexico"
    def __init__(self,val1,val2):
        self.modelo = val1
        self.serie  = val2
        self.marca  = "Ford"
        self.color  = "Rojo"
        print('Nuevo vehiculo creado')
        print('Modelo: ',self.modelo)
        print('Marca: ',self.serie)
```

```
myAuto = vehiculo(2020,"X25FG")
```

```
In [12]: myAuto.fabrica
Out[12]: 'Puebla'

In [13]: myAuto.pais
Out[13]: 'Mexico'
```

ITAM

Encapsulamiento

Encapsulamiento

- La encapsulación se refiere a impedir el acceso a determinados métodos y atributos de los objetos estableciendo así qué puede y no puede utilizarse desde fuera de la clase

- Otros lenguajes (e.g. Java) tiene la opción de definir atributos y métodos públicos y privados

- Python utliza _ _

Getters & Setters

- Controlar el acceso a atributos
- Uso de __ antes de definir atributos
- Obtener valor de atributos con métodos **get**
- Asignar valor a atributos con métodos **set**

Getters & Setters

- Ejemplo: Agregando __ a atributos de clase y objeto

```
class vehiculo:
    __fabrica = "Puebla"
    __pais = "Mexico"
    def __init__(self,val1,val2):
        self.__modelo = val1
        self.__serie  = val2
        self.__marca  = "Ford"
        self.__color  = "Rojo"
        print('Nuevo vehiculo creado')
        print('Modelo: ',self.__modelo)
        print('Marca: ',self.__serie)
```

```
myAuto = vehiculo(2020,"X25FG")
```

```
In [16]: myAuto.pais
Traceback (most recent call last):

  File "<ipython-input-16-1634be37a72c>", line 1, in <module>
    myAuto.pais

AttributeError: 'vehiculo' object has no attribute 'pais'
```

Getters & Setters

- Ejemplo: Definiendo getters

```
class vehiculo:
    __fabrica = "Puebla"
    __pais = "Mexico"
    def __init__(self,val1,val2):
        self.__modelo = val1
        self.__serie  = val2
        self.__marca  = "Ford"
        self.__color  = "Rojo"
        print('Nuevo vehiculo creado')
        print('Modelo: ',self.__modelo)
        print('Marca: ',self.__serie)

    def getPais(self):
        return self.__pais

    def getFabrica(self):
        return self.__fabrica

    def getModelo(self):
        return self.__modelo

myAuto = vehiculo(2020,"X25FG")
```

```
In [20]: myAuto.getPais()
Out[20]: 'Mexico'

In [21]: myAuto.getFabrica()
Out[21]: 'Puebla'

In [22]: myAuto.getModelo()
Out[22]: 2020
```

Getters & Setters

- Ejemplo: Definiendo setters

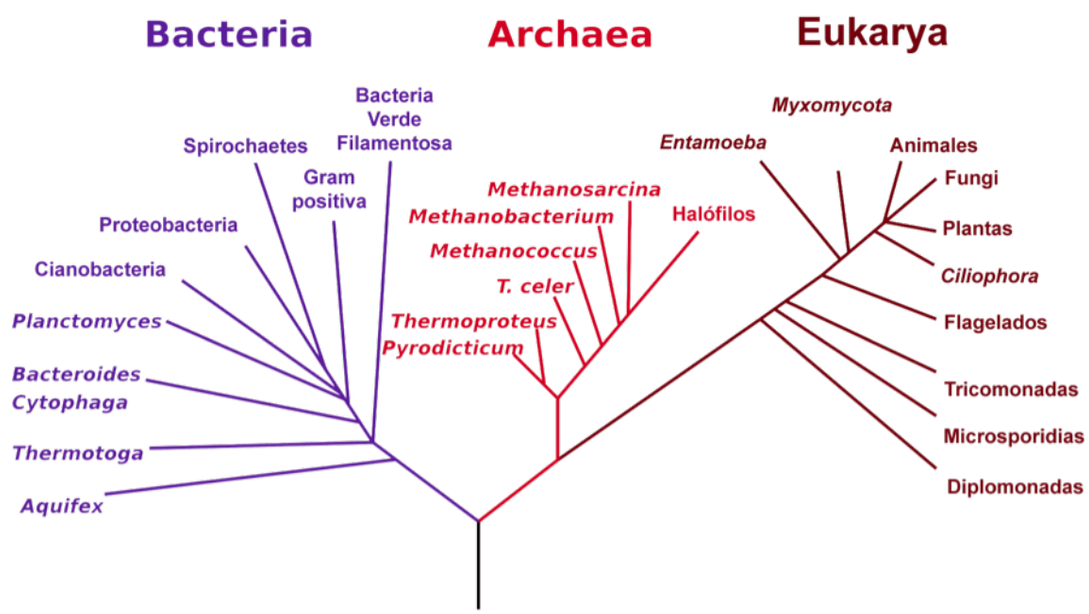
```
class vehiculo:
    __fabrica = "Puebla"
    __pais = "Mexico"
    def __init__(self,val1,val2):
        self.__modelo = val1
        self.__serie  = val2
        self.__marca  = "Ford"
        self.__color  = "Rojo"
        print('Nuevo vehiculo creado')
        print('Modelo: ',self.__modelo)
        print('Marca: ',self.__serie)

    def getPais(self):
        return self.__pais

    def getFabrica(self):
        return self.__fabrica

    def getModelo(self):
        return self.__modelo

    def setPais(self,pais):
        self.__pais = pais
        print("Definición de pais efectuada correctamente")

    def setModelo(self,modelo):
        self.__modelo = modelo
        print("Definición de modelo exitosa")
```

```
Nuevo vehiculo creado
Modelo:  2020
Marca:  X25FG
Vehiculo con # de serie X25FG destruido

In [24]: myAuto.getModelo()
Out[24]: 2020

In [25]: myAuto.setModelo(2021)
Definición de modelo exitosa

In [26]: myAuto.getModelo()
Out[26]: 2021
```

Herencia

Herencia

- Idea general
  - Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación
- Herencia sencilla
  - Una clase hereda atributos y métodos de una clase “superior”
- Herencia multiple
  - Una clase hereda atributos y métodos de más de una clase



Herencia

- Ejemplo: Herencia simple

```
class autoSedan(vehiculo):
    __puertas = 4
    def __init__(self,val1):
        self.__precio = val1

    def getPrecio(self):
        return self.__precio

    def setPrecio(self,precio):
        self.__precio = precio
        print("Definición de precio exitosa")

myAuto = vehiculo(2020,"X25FG")
myNuevoAuto = autoSedan(100000)
```

```
In [52]: myNuevoAuto?
Type:        autoSedan
String form: <__main__.autoSedan object at 0x115787b38>
Docstring:   <no docstring>

In [53]: myNuevoAuto.getPrecio()
Out[53]: 100000

In [54]: myNuevoAuto.getPais()
Out[54]: 'Mexico'
```

Herencia

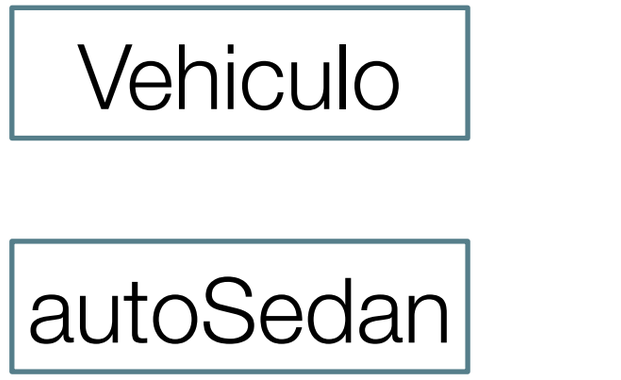
- Ejemplo: Herencia simple

```
In [55]: myNuevoAuto.getModelo()
Traceback (most recent call last):

  File "<ipython-input-55-43c5ee55d6c8>", line 1, in <module>
    myNuevoAuto.getModelo()

  File "/Users/ivan/Documents/ITAM/Cursos/2020/AyP-2020-Enero-Mayo/Ejemplos/oop_ej_02.py", line 28, in
getModelo
    return self.__modelo

AttributeError: 'autoSedan' object has no attribute '_vehiculo__modelo'
```

Herencia

- Ejemplo: Herencia simple

```
class autoPrototipo(autoSedan):
    __turbinas = 2
    def __init__(self,var1):
        self.__patente = var1

    def getPatente(self):
        return self.__patente

myAuto = vehiculo(2020,"X25FG")
myNuevoAuto = autoSedan(100000)
myPrototipo = autoPrototipo("ABC2345")
```

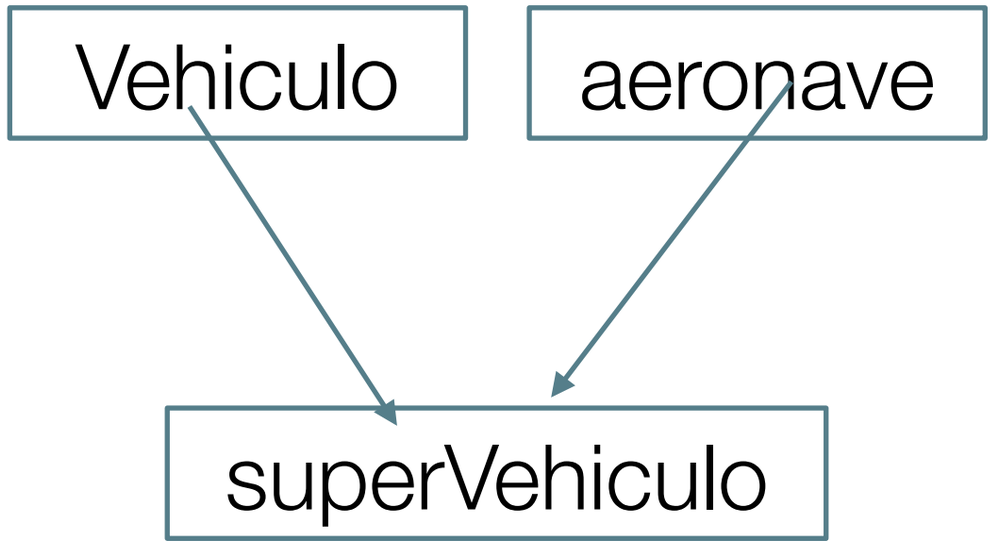



```
In [59]: myPrototipo?
Type:        autoPrototipo
String form: <__main__.autoPrototipo object at 0x1156c6cf8>
Docstring:   <no docstring>

In [60]: myPrototipo.getPatente()
Out[60]: 'ABC2345'
```

Herencia

- Ejemplo: Herencia multiple

```
class aeronave:
    __alas = 2
    def getAlas(self):
        return self.__alas

class superVehiculo(vehiculo,aeronave):
    __turbinas = 4
    def getTurbinas(self):
        return self.__turbinas

myAuto = vehiculo(2020,"X25FG")
myNuevoAuto = autoSedan(100000)
myPrototipo = autoPrototipo("ABC2345")
myNuevoVehiculo = superVehiculo(2050,"AAA")
```




```
In [63]: myNuevoVehiculo?
Type:        superVehiculo
String form: <__main__.superVehiculo object at 0x115787b70>
Docstring:   <no docstring>

In [64]: myNuevoVehiculo.getModelo()
Out[64]: 2050

In [65]: myNuevoVehiculo.getPais()
Out[65]: 'Mexico'

In [66]: myNuevoVehiculo.getTurbinas()
Out[66]: 4
```

ITAM