PROGRAMACIÓN II - UNIDAD 2

Ing. Gastón Weingand (gaston.weingand@uai.edu.ar)

Agenda

1. Desarrollo y utilización de clases y objetos. Variables de clases e instancias.
2. Constructores y destructores.
3. Encapsulamiento.
4. Herencia simple y herencia múltiple.
5. Polimorfismo.
6. Clases de “nuevo-estilo”.
7. Métodos especiales.
8. Tipos en Python.

Agenda

1. Desarrollo y utilización de clases y objetos. Variables de clases e instancias.
2. Constructores y destructores.
3. Encapsulamiento.
4. Herencia simple y herencia múltiple.
5. Polimorfismo.
6. Clases de “nuevo-estilo”.
7. Métodos especiales.
8. Tipos en Python.

1- Clases y objetos

En Python las clases se definen mediante la palabra clave class seguida del nombre de la clase, dos puntos (:) y a continuación, indentado, el cuerpo de la clase.

Como en el caso de las funciones, si la primera línea del cuerpo se trata de una cadena de texto, esta será la cadena de documentación de la clase o docstring.

1- Clases y objetos - Declaración e instanciación

Declaración

```python
class Coche:
    def __init__(self, gasolina):
        self.__gasolina = gasolina
        print("Tenemos", gasolina, "litros")
```

Instanciación

```
mi_auto = Coche(5)
```

1- Clases y objetos – Miembros de clase

Atributos de clase

```python
class Coche:

    marca =  None #Atributo de clase público
    __modelo = None #Atributo de clase privado
```

Método de clase

```python
@classmethod
def imprimir(self):
    print("Yo soy un coche", self.marca)
```

1- Clases y objetos – Miembros de instancia

Atributos de instancia

```python
def __init__(self, gasolina, marca):
    self.marca = marca #Atributo de instancia público
    self.__gasolina = gasolina #Atributo de instancia privado
    print("Tenemos", gasolina, "litros")
```

Método de instancia

```python
def arrancar(self):
    if self.__gasolina > 0:
        print("Arranca")
    else:
        print("No arranca")
```

1- Clases y objetos – Miembros estático

Los métodos estáticos requieren del decorador **@staticmethod** para indicarle a python que se trata del mismo. Este tipo de método nos permite llamar una función elegantemente dentro de una clase sin que esté ligada a la clase misma ni a la instancia

```
@staticmethod
def funcion_sin_relacion_con_la_clase(origen, destino, kms): #Definimos el metodo
    print ("El automovil irá desde", origen, "hasta", destino, "recorriendo", kms, "kms")
```

Agenda

1. Desarrollo y utilización de clases y objetos. Variables de clases e instancias.
2. Constructores y destructores.
3. Encapsulamiento.
4. Herencia simple y herencia múltiple.
5. Polimorfismo.
6. Clases de “nuevo-estilo”.
7. Métodos especiales.
8. Tipos en Python.

2- Constructor y destructor

Definición

```
def __init__(self, gasolina): #Constructor
    self.__gasolina = gasolina
    print("Tenemos", gasolina, "litros")

def __del__(self): #Destructor
    print ("Eliminando objeto...")
```

Uso

```
auto2 = Coche(10)
del auto2
```

Agenda

1. Desarrollo y utilización de clases y objetos. Variables de clases e instancias.
2. Constructores y destructores.
3. Encapsulamiento.
4. Herencia simple y herencia múltiple.
5. Polimorfismo.
6. Clases de “nuevo-estilo”.
7. Métodos especiales.
8. Tipos en Python.

3- Encapsulamiento -> Propiedades

```python
def getGasolina(self):
    return self.__gasolina

def setGasolina(self, gasolina):
    if gasolina > 0:
        self.__gasolina = gasolina
    else:
        print("No se puede cargar menos o igual a 0 de combustible")

gasolina = property(getGasolina, setGasolina)
```

3- Name mangling

Name mangling: Los miembros privados no son verdaderamente privados…

```
class Ejemplo:
    def publico(self):
        print ("Publico")
    def __privado(self):
        print ("Privado")

ej = Ejemplo()
ej.publico()
ej._Ejemplo__privado()
```

Ejemplo 1 - DEFINICIÓN DE CLASE

```python
class Coche:

    marca =  None #Atributo de clase público
    __modelo = None #Atributo de clase privado

    def __init__(self, gasolina): #Constructor
        self.__gasolina = gasolina
        print("Tenemos", gasolina, "litros")

    def __del__(self): #Destructor
        print ("Eliminando objeto...")

    @classmethod
    def setModelo(self, modelo):
        self.__modelo = modelo
```

Ejemplo 1 - DEFINICIÓN DE CLASE

```
def getGasolina(self):
    return self.__gasolina

def setGasolina(self, gasolina):
    if gasolina > 0:
        self.__gasolina = gasolina
    else:
        print("No se puede cargar menos o igual a 0 de combustible")

gasolina = property(getGasolina, setGasolina)

def arrancar(self):
    if self.__gasolina > 0:
        print("Arranca")
    else:
        print("No arranca")

def conducir(self):
    if self.__gasolina > 0:
        self.__gasolina -= 1
        print("Quedan", self.__gasolina, "litros")
    else:
        print("No se mueve")
```

Ejemplo 1 - DEFINICIÓN DE CLASE

```
@classmethod
def imprimir(self):
    print("Yo soy un coche", self.marca, "modelo", self.__modelo)

@staticmethod
def funcion_sin_relacion_con_la_clase(origen, destino, kms): #Definimos el metodo
    print ("El automovil irá desde", origen, "hasta", destino, "recorriendo", kms, "kms")
```

Ejemplo 1 - USO

```
auto1 = Coche(5) #Instancio un coche
Coche.marca = "Fiat" #Modifico atributo público de clase
Coche.setModelo("Argo") #Modifico atributo privado de clase (setter)
auto1.arrancar() #Arrancar el vehículo, método de instancia
auto1.conducir() #Conducir el vehículo, método de instancia
#auto1.setGasolina(10)
auto1.setGasolina(0) #Método setter tradicional
auto1.gasolina = 10 #Seteo por property
print(auto1.gasolina) #Traigo la gasolina por property
print(auto1.getGasolina()) #Traigo la gasolina por getter tradicional
auto1.conducir()
auto1.conducir()
Coche.imprimir() #Método de clase -> LLamo directo a la clase
Coche.funcion_sin_relacion_con_la_clase("bs as", "rosario", 45) #Método estático -> Sin relación interna
del auto1 #Destruyo el objeto
```

Ejemplo 2 - Composición

```python
#Composición
class Salary:
    def __init__(self, pay):
        self.__pay=pay

    def get_total(self):
        return (self.__pay * 12)

class Employee:
    def __init__(self, pay, bonus):
        self.__pay=pay
        self.__bonus=bonus
        self.__obj_salary=Salary(self.__pay) #Compongo a la clase Salario

    def annual_salary(self):
        return "Total: "  +  str(self.__obj_salary.get_total() + self.__bonus)


obj_emp=Employee(100, 10)
print (obj_emp.annual_salary())
```

Ejemplo 3 - Agregación

```python
#Agregación
class Salary:
    def __init__(self, pay):
        self.__pay=pay

    def get_total(self):
        return (self.__pay * 12)

class Employee:
    def __init__(self, pay, bonus):
        self.__pay=pay #Paso directamente el objeto Salario en el constructor
        self.__bonus=bonus
        #self.__obj_salary=Salary(self.__pay)

    def annual_salary(self):
        return "Total: "  +  str(self.__pay.get_total() + self.__bonus)

obj_salary=Salary(100) #Creo el objeto salario
obj_emp=Employee(obj_salary, 10) #Lo paso como parámetro al constructor a Empleado
print (obj_emp.annual_salary())
```