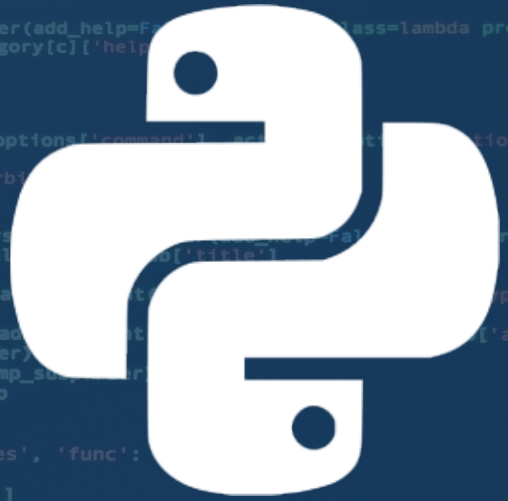

ORIENTACIÓN A OBJETOS

PROGRAMACIÓN III

Abdel G. Martínez L.

CREAR CLASES

```
class Empleado:
   'Clase base para todos los empleados'
   conteo = 0

   def __init__(self, nombre, salario):
      self.nombre = nombre
      self.salario = salario
      Empleado.conteo += 1

   def mostrarConteo(self):
     print("Empleados totales: " + str(Empleado.conteo))

   def mostrarEmpleado(self):
      print("Nombre : "+ self.nombre + ", Salario: "+ self.salario)
```

CREAR OBJETOS

```
"Crea el primer objeto de la clase Empleado"
emp1 = Empleado(”Analida", 2500)
"Crea el segundo objeto de la clase Empleado"
emp2 = Empleado(”Clarissa", 4500)
```

ACCEDER ATRIBUTOS

```
emp1.mostrarEmpleado()
emp2. mostrarEmpleado()
print(“Empleados Totales: + str(Employee.conteo)”)
```

ATRIBUTOS PROPIOS DE CLASES

- **__dict__**: Diccionario con el *namespace* de la clase.
- **__doc__**: Documentación de la clase.
- **__name__**: Nombre de la clase.
- **__module__**: Nombre del módulo donde la clase es definida.

DESTRUCCIÓN DE OBJETOS (GC)

```
class Libro:
    def __init__( self, x=0, y=0):
        self.x = x
        self.y = y
    def __del__(self):
        nombre_clase = self.__class__.__name__
        print(nombre_clase, ”destruido”)

lb1= Libro()
lb2 = lb1
del lb1
del lb2
```

HERENCIA

```
class Parent:        # define parent class
   parentAttr = 100
   def __init__(self):
      print("Calling parent constructor”)

   def parentMethod(self):
      print('Calling parent method’)

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print("Parent attribute :", Parent.parentAttr)
```

HERENCIA

```
class Child(Parent): # define child class
   def __init__(self):
      print "Calling child constructor"

   def childMethod(self):
      print 'Calling child method'

c = Child()           # instance of child
c.childMethod()       # child calls its method
c.parentMethod()      # calls parent's method
c.setAttr(200)        # again call parent's method
c.getAttr()           # again call parent's method
```

SOBREESCRIBIR MÉTODOS

```
class Parent:          # define parent class
   def myMethod(self):
      print 'Calling parent method'

class Child(Parent): # define child class
   def myMethod(self):
      print 'Calling child method'

c = Child()            # instance of child
c.myMethod()           # child calls overridden method
```

SOBRECARGAR MÉTODOS

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

ABSTRACCIÓN

```
class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```