Programación Orientada a Objetos

Conceptos

- Objeto
- Clase
- Atributos
- Métodos
- Instancia
- Herencia

Objeto

- Entidad que agrupa un estado determinado y una funcionalidad relacionada.
- El estado del objeto se define a través de variables llamadas ***Atributos***.
- La funcionalidad se modela a través de funciones (***Métodos***) que alteran el estado.

Ejemplo de objeto Estrella

- Atributos:
  - Nombre
  - Temperatura
  - Magnitud Visual
  - Tipo Espectral
  - Diámetro

Clase

- Plantilla genérica que define las características, el concepto abstracto (como pensar en lo que es una estrella, pero no en ninguna en particular).
- La clase define los atributos que todas las estrellas tienen en común, y a partir de la clase, pueden “crearse” estrellas individuales.
- También puede pensarse en las clases como los planos de un edificio.

Instancia

- Una instancia es una entidad construida a partir de la definición de una clase.
- Cada instancia tiene valores propios para los atributos, que distinguen una de otra.

Instancias de Estrella

Nombre	Sol
Temperatura	5700
Magnitud Visual	-26
Tipo Espectral	G2
Diámetro	1.39^6 km

Nombre	Betelgeuse
Temperatura	3140
Magnitud Visual	0.42
Tipo Espectral	M2
Diámetro	1000 Rs

Nombre	Sirio
Temperatura	9940
Magnitud Visual	-1.47
Tipo Espectral	A1
Diámetro	1.71 Rs

Nombre	Polaris
Temperatura	6000
Magnitud Visual	1.98
Tipo Espectral	F7
Diámetro	46 Rs

Constructor

- El ***constructor*** es un método especial de las clases que se usa para crear nuevas instancias.
- Es la forma de crear variables que almanecen un objeto de una clase.
- Se le pueden pasar valores que inicialicen los atributos al momento de crear la clase.

Definición de clases en Python

```python
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import math

class Estrella(object):
    """Definición de la clase Estrella para modelar
    objetos astrofísicos

    """

    def __init__(self, name, temp, vmag, espectral_type, d):
        """Constructor de la clase estrella
        Inicializa los diferentes atributos

        """
        self.nombre = name
        self.temperatura = temp
        self.magnitud_visual = vmag
        self.tipo_espectral = espectral_type
        self.diametro = d

    def calcular_area(self):
        """Calcula el área de la estrella basado en su diametro"""

        radio = self.diametro / 2
        return 4.0 * math.pi * radio * radio

    def calcular_brillo(self):
        """Calcula un valor de brillo ficticio a partir de la temperatura de la estrella
        y el area de su superficie

        """

        return self.temperatura * self.calcular_area()
```

Creando instancia de estrella

```
In [6]: from estrella import Estrella    #Importamos el modulo con la clase

In [7]: sol = Estrella("sol", 5700, -26, "G2", 1.4E7)

In [8]: sirio = Estrella("sirio", 9940, -1.47, "A1", (sol.diametro/2.0) * 1.71)

In [9]: sol.calcular_area()
Out[9]: 615752160103599.5

In [10]: sirio.calcular_area()
Out[10]: 450130222839733.8

In [11]: sol.calcular_brillo()
Out[11]: 3.509787312590517e+18

In [12]: sirio.calcular_brillo()
Out[12]: 4.474294415026954e+18
```

Print del objeto

```
In [5]: print sol
<estrella.Estrella object at 0x1b12450>

In [6]
<estre
```

```
In [5]: print sol
Estrella: sol - VMag: -26 - Temp: 5700

In [6]: print sirio
Estrella: sirio - VMag: -1.47 - Temp: 9940
```

```
 6 class Estrella(object):
 7     """Definición de la clase Estrella para modelar
 8     objetos astrofísicos
 9
10     """
11
12     def __init__(self, name, temp, vmag, espectral_type, d):
13         """Constructor de la clase estrella
14         Inicializa los diferentes atributos
15
16         """
17         self.nombre = name
18         self.temperatura = temp
19         self.magnitud_visual = vmag
20         self.tipo_espectral = espectral_type
21         self.diametro = d
22
23     def __str__(self):
24         return "Estrella: %s - VMag: %s - Temp: %s" % (self.nombre, self.magnitud_visual, self.temperatura)
```

Herencia

- Una forma de reutilizar código.
- Permite crear clases basadas en otras ya existentes.
- Crear especializaciones
- Ejemplo: crear un tipo de estrella GiganteRoja que “herede” las características generales de cualquier estrella.

```python
class GiganteRoja(Estrella):
    """Definición de la clase GiganteRoja
    a partir de la clase Estrella

    """

    def __init__(self, name, vmag):
        # Vamos a asignarle valores por defecto basado en lo que conocemos sobre las gigantes rojas
        Estrella.__init__(self, name, 3000, vmag, "M2", 7E5 * 1000)

    def expandir(self, factor):
        # Metodo expandir que aumenta el radio baja la temperatura como consecuencia
        self.diametro += (self.diametro * factor)
        self.temperatura -= self.temperatura * factor
```

```
In [2]: from estrella import Estrella, GiganteRoja

In [3]: sol = Estrella("sol", 5700, -26, "G2", 1.4E7)

In [4]: gigante = GiganteRoja("betelgeuse", 0.4)

In [5]: print sol
Estrella: sol - VMag: -26 - Temp: 5700

In [6]: sol.calcular_area()
Out[6]: 615752160103599.5

In [7]: gigante.calcular_area()
Out[7]: 1.5393804002589985e+18

In [8]: sol.calcular_brillo()
Out[8]: 3.509787312590517e+18

In [9]: gigante.calcular_brillo()
Out[9]: 4.618141200776996e+21
```

```
In [11]: gigante.tipo_espectral
Out[11]: 'M2'

In [12]: sol.tipo_espectral
Out[12]: 'G2'

In [13]: sol.magnitud_visual
Out[13]: -26

In [14]: gigante.magnitud_visual
Out[14]: 0.4
```

Herencia

```
In [23]: sol.
sol.calcular_area    sol.magnitud_visual  sol.tipo_espectral
sol.calcular_brillo  sol.nombre
sol.diametro         sol.temperatura

In [23]: gigante.
gigante.calcular_area    gigante.expandir          gigante.temperatura
gigante.calcular_brillo  gigante.magnitud_visual   gigante.tipo_espectral
gigante.diametro         gigante.nombre
```

```
In [29]: print gigante, gigante.diametro
Estrella: betelgeuse - VMag: 0.4 - Temp: 3000 700000000.0

In [30]: gigante.expandir(0.25)

In [31]: print gigante, gigante.diametro
Estrella: betelgeuse - VMag: 0.4 - Temp: 2250.0 875000000.0

In [32]: sol.expandir(0.25)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/home/jeudy/Proyectos/UCR/intropython0415/<ipython-input-32-0fa6323d1958> in <mo
dule>()
----> 1 sol.expandir(0.25)

AttributeError: 'Estrella' object has no attribute 'expandir'
```

Ejemplo: Círculo

- Un círculo se define por su radio (atributo)
- Agreguemos 2 métodos básicos: circunferencia y área.

```python
# -*- coding: UTF-8 -*-

import math

"""Clase que define a la figura geométricula Círculo
El constructor recibe el valor del radio como parámetro.
Los métodos que tiene son: area y circunferencia

"""

class Circulo(object):

    # Constructor
    def __init__(self, r):
        self.radio = r

    # Metodo especial para imprimir objeto amigablemente
    def __str__(self):
        return "Circulo de radio: %s " % (self.radio)

    # Area del circulo
    def area(self):
        return math.pi * self.radio**2

    def circunferencia(self):
        return 2.0 * math.pi * self.radio
```

```
In [2]: from figuras import Circulo

In [3]: c1 = Circulo(10)

In [4]: print c1
Circulo de radio: 10

In [5]: print c1.circunferencia()
62.8318530718

In [6]: print c1.area()
314.159265359
```

Ejemplo: Cuenta bancaria

- Atributos:
  - Saldo (inicial será cero)
  - Número
  - Descripción
- Métodos:
  - Constructor (numero y descripción)
  - Depositar (monto, modifica saldo)
  - Retirar (monto, modifica saldo, valida disponibilidad)

```
In [2]: from cuenta import CuentaBancaria

In [3]: ahorros = CuentaBancaria(100001, "Cuenta de Ahorros")

In [4]: print ahorros
Cuenta #100001 - Cuenta de Ahorros con un saldo: 0.0

In [5]: ahorros.depositar(1000)

In [6]: ahorros.depositar(5000)

In [7]: print ahorros
Cuenta #100001 - Cuenta de Ahorros con un saldo: 6000.0

In [8]: ahorros.retirar(4500)

In [9]: print ahorros
Cuenta #100001 - Cuenta de Ahorros con un saldo: 1500.0

In [10]: ahorros.retirar(10000)
Fondos insuficientes.

In [11]: print ahorros
Cuenta #100001 - Cuenta de Ahorros con un saldo: 1500.0
```

```
In [14]: corriente = CuentaBancaria(200000, "Cuenta corriente")

In [15]: corriente.depositar(5000000)

In [16]: print ahorros
Cuenta #100001 - Cuenta de Ahorros con un saldo: 1500.0

In [17]: print corriente
Cuenta #200000 - Cuenta corriente con un saldo: 5000000.0
```

```
Clase para modelar una cuenta bancaria.
Atributos:
        numero: numero de cuenta (integer)
        descripcion (string)
        monto (float)

Métodos:
    depositar: recibe un monto, aumenta el saldo con ese monto
    retirar: recibe un monto, revisa si hay saldo suficiente y resta
"""

class CuentaBancaria(object):

    def __init__(self, numero, descripcion):
        self.numero = numero
        self.description = descripcion
        self.saldo = 0.0

    def __str__(self):
        return "Cuenta #%s - %s con un saldo: %s" % (self.numero, self.description, self.saldo)

    def depositar(self, monto):
        self.saldo += monto

    def retirar(self, monto):
        if self.saldo - monto < 0:
            print "Fondos insuficientes."
        else:
            self.saldo -= monto
```

Ejemplo: Polinomio

- Vamos a representar un polinomio de una variable como la lista de los coeficientes.
- El grado del polinomio será igual al tamaño de la lista – 1
- Ejemplo:
  - $2x^2 + 3x + 5$

  se representará por medio de la lista de coeficientes: [5, 3, 2] (empieza por el exponente más pequeño)

Ejemplo: Polinomio

- Tendremos un método evaluar, que reciba el valor de x, y devolverá el valor de evaluar ese polinomio en x.
- Tendremos otro método derivar, que devuelve como resultado la primera derivada del polinomio (otro polinomio!)
- Ejemplo: $2x^2 + 3x + 5 \rightarrow 4x + 3 \rightarrow [3, 4]$

```
In [2]: from polinomio import Polinomio

In [3]: p1 = Polinomio(5,3,2)

In [4]: print p1
5x^0 + 3x^1 + 2x^2

In [5]: print p1.evaluar(10)
235

In [6]: derivada = p1.derivar()

In [7]: print derivada
3x^0 + 4x^1

In [8]: derivada.evaluar(10)
Out[8]: 43

In [9]: segunda_derivada = p1.derivar().derivar()

In [10]: print segunda_derivada
4x^0

In [11]: print segunda_derivada.evaluar(10)
4
```

```python
class Polinomio(object):

    def __init__(self, *lista_coeficientes):
        self.coeficientes = lista_coeficientes

    def __str__(self):
        cadena = ""
        for i, x in enumerate(self.coeficientes):
            cadena += "%sx^%d + " % (x, i)
        return cadena[:-2]      # remueve el + vacio al final

    def evaluar(self, x):
        resultado = 0
        for i, coef in enumerate(self.coeficientes):
            resultado += coef * x**i
        return resultado

    def derivar(self):
        tmp_coeficientes = self.coeficientes[1:]    # Desechamos el primer coeficiente (exponente 0)
        nuevos_coeficientes = []
        for i, x in enumerate(tmp_coeficientes):
            nuevos_coeficientes.append(x * (i+1))

        return Polinomio(*nuevos_coeficientes)  #El * es para llamar el constructor con N elementos, no con una lista como unico parametro
```

Ejercicio (Herencia)

- Cree una clase Figura, que reciba en su constructor una lista con la medida de los lados.
- Cree el metodo especial __str__
- Agreguele un método perimetro que devuelva la suma de sus lados.

```
In [2]: from figuras import Figura

In [3]: tri = Figura([10,10,10])

In [4]: print tri
Figura: lados = 10, 10, 10,

In [5]: tri.perimetro()
Out[5]: 30

In [6]: cuadro = Figura([20,20,20,20])

In [7]: print cuadro
Figura: lados = 20, 20, 20, 20,
```

Ejercicio (Herencia)

- Cree una clase Cuadrado, que herede de Figura, reciba como parámetro del constructor solo el valor de 1 de sus lados.
- Inicialice la lista para llamar al constructor de la clase padre.
- Guarde el lado como atributo de la clase.
- Agregue un método area que calcule el area del cuadrado.
- Sobreescriba el método especial __str__ para que muestre: “Cuadrado de lado: x”

```
In [2]: from figuras import Figura, Cuadrado

In [3]: cu = Cuadrado(20)

In [4]: print cu
Cuadrado de lado 20

In [5]: cu.perimetro()
Out[5]: 80

In [6]: cu.area()
Out[6]: 400
```