PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC1103- INTRODUCCIÓN A LA PROGRAMACIÓN

Laboratorio 5
Programación Orientada a Objetos

Segundo semestre, 2015

Objetivo

Este laboratorio es para que practiques con Programación Orientada a Objetos.

1. Programación Orientada a Objetos

Como habrás visto en clases, hay múltiples conceptos nuevos en la programación orientada a objetos (clases, métodos, atributos, sobrecarga, etc.). En este laboratorio escribirás programas con orientación a objetos para que de a poco te vayas familiarizando con el concepto. Partamos con un pequeño ejemplo, el cual desglosaremos para ver su funcionamiento.

```
class Rectangulo:
    def __init__(self, a=1, b=2):
        self.a = a
        self.b = b
    def area(self):
        return self.a*self.b
    def __str__(self):
        return "Rectangulo de lados " + str(self.a) + " y " + str(self.b)
```

Este código define un **objeto** de **clase** `Rectangulo`. Para decirle a Python que estamos definiendo un objeto, usamos la instrucción `class`. La primera línea,

```
class Rectangulo:
```

define que los códigos siguientes definirán al objeto `Rectangulo`. Nota que son necesarios los dos puntos después del nombre de la clase. Después, hay un **método** (función) que tiene un nombre que verás en gran parte de las definiciones de objetos: la función `__init__`, que es llamada el *constructor*:

```
    def __init__(self, a=1, b=2):
        self.a = a
        self.b = b
```

En el ejemplo, la función recibe las dimensiones de los lados del rectángulo (con parámetros por defecto de 1 y 2), y los guarda en una variable especial llamada `self`. Esta variable debes usarla como argumento en todos los métodos que definas dentro del objeto. Dentro de la función `__init__` se definen los **atributos** del

objeto `Rectangulo`, que en este caso son las dimensiones de cada lado. También podría agregarse como atributos, por ejemplo, el color, el uso del rectángulo, la fecha de creación, y en general cualquier atributo que sea relevante del objeto que estás definiendo.

Luego, tenemos el método `area`, que es una función que calcula el área del objeto `Rectangulo` y retorna su valor:

```
    def area(self):
        return self.a*self.b
```

Como puedes notar, el método `area` recibe como parámetro a la variable `self`. Esto le permite acceder a los valores de los lados del rectángulo `self.a` y `self.b`. Si necesitaras de otro valor como argumento, por ejemplo si quisieras escalar el área en un factor, bastaría con agregarlo después de la variable `self`, separado por una coma. Como puedes ver, el valor de retorno del método `area` es la multiplicación de los lados.

Por último, para terminar el ejemplo, vemos un método de nombre `__str__`:

```
    def __str__(self):
        return "Rectangulo de lados " +  str(self.a) + " y " +  str(self.b)
```

Esta función realiza una **sobrecarga** de la función `print()`, es decir, cada vez que ejecutemos la función `print()` usando como argumento un objeto de la clase `Rectangulo`, se ejecutará internamente la función `__str__`, y su valor de retorno será entregado como argumento a `print()`. Para que quede todo más claro, veamos un ejemplo de uso del objeto:

```
>>> rect = Rectangulo(2,3)
>>> area_rect = rect.area()
>>> print(area_rect)
6
>>> print(rect)
Rectangulo de lados 2 y 3
```

Del ejemplo, podemos inferir ciertas cosas interesantes:

- Para definir un objeto, usamos el nombre de la clase, y entre paréntesis ponemos el valor de los atributos que queremos que tenga el objeto, en este caso definimos una variable (o **instancia** de la clase) llamada `rect`, y para definirla usando lados 2 y 3 usamos `Rectangulo(2,3)`.
- Para usar los métodos del objeto creado, usamos el nombre del objeto (`rect` en el ejemplo), seguido de un punto, y seguido por el nombre del método, como en `rect.area()` que nos entrega el área del rectángulo.
- La variable `self` es interna a los métodos del objeto, por lo que al usar la notación `<objeto>.<metodo>(args)` sólo se definen los argumentos `args` adicionales a `self`. Por esto mismo, `rect.area()` no necesita de argumentos. Recuerda que las funciones siempre son llamadas con argumentos, y si no hay argumentos, igual deben ponerse los paréntesis al final.
- Puedes pensar en una clase como un tipo de dato creado por ti, que tiene formas de interactuar con otros datos mediante métodos. Al principio cuesta un poco digerir todo lo involucrado con clases, métodos, atributos, sobrecarga, instancias, etc., pero una vez lo entiendas bien verás que es un paradigma muy poderoso y práctico.

Dentro de los argumentos de los métodos puedes poner incluso otras instancias de la clase. También retornar nuevas instancias de la misma clase en los métodos. Analiza el siguiente ejemplo resuelto para ver estas características.

1.1. Ejemplo resuelto

Programa una clase `Hora`, que permita lo siguiente:

- Definir la hora actual y el huso horario.
- Devolver un objeto `Hora` en otro huso horario.
- Comparar con otra hora y retornar `True` si la hora a comparar es más tarde, o `False` si es igual o más temprano.
- Imprimir en pantalla, por ejemplo, "La hora definida es 14:30 GMT -3"

Solución

```
class Hora:

    def __init__(self,horas=0, minutos=0, GMT = -3):
        # Atributos: horas, minutos, y huso horario
        self.horas = horas
        self.minutos = minutos
        self.GMT = GMT

    def otroHuso(self, nuevo_GMT):
        # defino nueva instancia de Hora en nueva_Hora, copiando todos los atributos
        nueva_Hora = Hora(self.horas, self.minutos, self.GMT)

        # calculo la hora en el nuevo huso horario
        nueva_Hora.horas = (nueva_Hora.horas +  nuevo_GMT - self.GMT)%24
        nueva_Hora.GMT = nuevo_GMT

        # retorno la nueva hora
        return nueva_Hora

    def compararHora(self, otraHora):
        # obtengo la hora en un huso horario comun
        hora1 = self.horas - self.GMT
        hora2 = otraHora.horas - otraHora.GMT
        if hora1 != hora2:
            return hora1 > hora2
        else:
            return self.minutos > otraHora.minutos

    def __str__(self):
        return "La hora definida es " + str(self.horas) + ":" + str(self.minutos) \
                + " GMT " + str(self.GMT)
```

Ahora usemos la clase definida:

```
>>> h1 = Hora(14,30) # usamos GMT -3
>>> h2 = h1.otroHuso(2) # Ej. Madrid con GMT +2 (horario de verano)
>>> print(h1)
La hora definida es 14:30 GMT -3
>>> print(h2)
La hora definida es 19:30 GMT 2
>>> h3 = Hora(15,10)
>>> resp = h1.compararHora(h3)
>>> print(resp)
False
```

2. Ejercicios

1. (Funciones) La ecuación general de la recta se escribe como $ax + by + c = 0$, donde a, b y c son parámetros, y (x, y) es un punto en el plano cartesiano. Mediante funciones, programa lo siguiente:

   a) Una función que reciba los parámetros (a, b, c), y las coordenadas de un punto (x, y), y retorne `True` si el punto pertenece a la recta y `False` si es que no percenece. Usa una tolerancia de 0.0001.

   b) Una función que reciba los parámetros (a, b, c), y un valor x, y entregue el valor de y tal que (x, y) pertenezca a la recta.

   c) Una función que reciba los parámetros (a, b, c), y un punto (x, y), y que retorne la distancia del punto a la recta $dist = \frac{|ax+by+c|}{\sqrt{a^2+b^2}}$

   d) Una función que reciba un punto (x, y), y que entregue la norma del vector $norm = \sqrt{x^2 + y^2}$.

   e) Una función que reciba dos puntos (x_1, y_1) y (x_2, y_2), y que retorne la distancia entre los dos puntos $dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

2. (Programación orientada a objetos) Programa una clase llamada `Fraccion`, cuyos atributos son `numerador` y `denominador`, y que permita realizar lo siguiente:

   a) Operaciones entre dos instancias de clase `Fracción`, que devuelve una nueva instancia de `Fracción`
      - `suma`
      - `resta`
      - `multiplicación`
      - `división`

   b) Métodos de la instancia
      - Imprimir la fracción en formato `<numerador>/<denominador>` al usar una instancia de `Fracción` dentro de la función `print()`.
      - Convertir la fracción a su mínima expresión, mediante un método `simplificar`.
      - Calcular el número real que representa a la fracción, mediante un método `tofloat`.

   Por ejemplo, un salida de tu programa podría ser la siguiente:

```
>>> f1 = Fraccion(2,3)
>>> f2 = Fraccion(1,2)
>>> f3 = f1.suma(f2)
>>> print(f1)
2/3
>>> print(f2)
1/2
>>> print(f3)
7/6
>>> realf3 = f3.tofloat()
>>> print(realf3)
1.1666666666666667
>>> f4 = Fraction(10,20)
>>> print(f4)
10/20
>>> f4.simplificar()
>>> print(f4)
1/2
```

3. (Programación orientada a objetos) Como en el Laboratorio 3, programa un mini-Blackjack numérico, usando números en vez de cartas, y jugado entre un usuario y el computador, esta vez usando programación orientada a objetos. Puedes reutilizar código del Laboratorio 3 si lo tienes disponible.

   El objetivo del juego es acercarse lo más posible a 21, sin pasarse. Gana el usuario si su número es mayor que el del computador. Los pasos del juego son:

   - El usuario inicia el juego con saldo de $1000.
   - El usuario apuesta una cantidad entre 1 y 100.
   - El computador le entrega un número entre 2 y 21.
   - El usuario puede pedir otro número entre 1 y 10 para aumentar el número inicial. Esto lo puede hacer varias veces. Si se pasa de 21, se le resta el dinero de la apuesta y el juego termina.
   - Si el usuario no quiere aumentar más el número, el computador imprime su número (entre 2 y 21). Si el número es mayor que 16, el computador no puede elevar más su número. Si el número es menor o igual a 16, el computador está obligado a generar un nuevo número (entre 1 y 10) y sumarlo al anterior, y repite el proceso mientras su número sea menor o igual a 16. Si el computador se pasa de 21, entrega al usuario como premio el doble del dinero de la apuesta. Si el número del computador es mayor o igual al del usuario, gana el computador.
   - El usuario puede optar al final seguir jugando o salir del programa.

   Como referencia para que uses programación orientada a objetos, las clases que debes definir, y algunos de sus atributos y métodos, son los siguientes:

   a) Una clase `Jugador_humano`:
      - Atributos: `dinero`, `numero_actual` y `apuesta_actual`.
      - Métodos: `apostar` y `recibir_numero`.

   b) Una clase `Jugador_computador`:
      - Atributos: `dinero` y `numero_actual`.
      - Métodos: `recibir_apuesta`, `repartir_numero_humano` y `repartir_numero_computador`.

c) Una clase `Blackjack`:

- Atributos: `humano` de clase `Jugador_humano`; y `computador`, de clase `Jugador_computador`
- Métodos: `iniciar_juego`, `declarar_ganador` y `repartir_dinero`

En los métodos que se requiera, valida los datos de entrada, por ejemplo que la apuesta sea entre 1 y 100.

Se muestra una posible salida del programa:

```
Tu saldo es 1000. Cuanto deseas apostar?: 100
Tu apuesta es 100. Tu numero es 14.
Deseas aumentar el numero? (si / no): si
Tu aumento es de 6. Tu nuevo numero es 20.
Deseas aumentar el numero? (si / no): no
El computador tiene el numero 12.
El computador aumenta el numero en 6. Ahora tiene el 18.
El computador no puede aumentar mas.
Gana usuario (20>18). Su nuevo saldo es 1100.
Deseas continuar jugando? (si / no): no
Gracias por jugar!!!
```

3. Bonus

Reformula el Ejercicio 1 usando programación orientada a objetos, crendo las clases `Punto2D` y `Recta2D`.