Guía de ejercicios POO - Clases y objetos - Parte 1

CONDICIONES DE ENTREGA:

- Archivo en formato .zip adjuntado en el classroom. Por cada ejercicio, debe haber un archivo con la función y otro con el test unitario. Si se utiliza Replit o algún editor de código en línea, se puede enviar un solo archivo por cada ejercicio
- La fecha de entrega estará indicada vía classroom

ACLARACIÓN GENERAL:

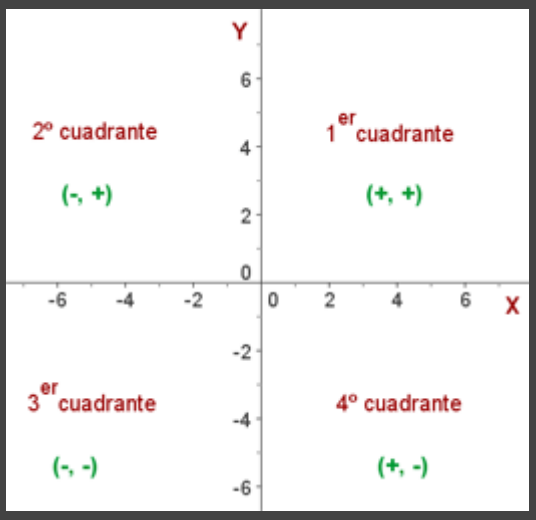
- No se debe utilizar la función input(). No es necesaria, ya que cada ejercicio contara con su propia Prueba Unitaria
- Cuando se pida la implementación del método __str__ recuerden que ese método devuelve un dato de tipo ***string***
- Cuando se pida la implementación del método __eq__ recuerden que ese método devuelve un dato de tipo ***bool***
- Se valorará más la aplicación del concepto de encapsulación (es decir, que aquellos atributos de los cuales necesitemos alterar su valor o simplemente consultarlo, posean sus propios métodos getter y setter)
- Los archivos de las pruebas unitarias serán provistos por los profesores, es decir, que el alumno no deberá escribirlos. Ahora bien, las pruebas unitarias ya están configuradas, es decir que, si la o las clases que implementa el alumno NO coinciden con los nombres de los métodos, el alumno tendrá que renombrar las clases y métodos del test unitario, o de sus propios archivos

1. Crear una clase **Rectángulo** que tenga
    a. Los atributos *base* y *altura*.
    b. Crear el constructor de la clase que reciba por argumento los valores de la base y la altura.
    c. Los métodos necesarios para:
        i. Calcular el área. La fórmula es base*altura
        ii. El perímetro. La formula es 2*base + 2*altura

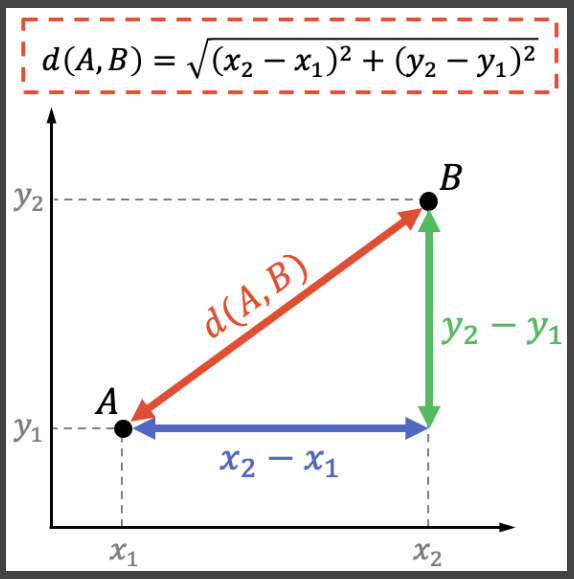
2. Crear una clase **Lampara** que tenga:
    a. Un *método* que prenda la lámpara
    b. Un *método* que apague la lámpara
    c. Un *método* que alterne el prendido y apagado de la lámpara (por ejemplo: si la lámpara estaba prendida, al ejecutarse el método pasará a estado apagada)
    d. Un *método* que retorne un string según su estado de prendida o apagada:
        i. Si estaba prendida, el *método* debe retornar “PRENDIDA”
        ii. Si estaba apagada, el *método* debe retornar “APAGADA”
    e. IMPORTANTE: analizar qué atributo debería tener nuestra clase, para poder implementar las funcionalidades solicitadas.

3. Crear una clase **Calculadora** que pueda realizar las acciones de *sumar*, *multiplicar*, *dividir* y *restar.* Todos los *métodos* de las operaciones poseen dos parámetros/argumentos de entrada (es decir, parámetros o argumentos que recibe el método)
IMPORTANTE: analizar si es necesario declarar atributos en la clase para resolver este problema.

4. Crear la *clase* **Punto** en el plano que contiene:
    a. 2 *atributos* 'x' e 'y' que representan las coordenadas del punto en el plano
    b. El constructor de la *clase* **Punto** que recibe por agumentos 'x' e 'y' que son las coordenadas
    c. Por defecto, inicializar los atributos 'x' e 'y' en el centro de coordenadas (es decir, 0, 0), a menos que reciba por argumento los valores en el constructor. Buscar como se inicializan los parámetros de un método si no se pasa por argumento ningún valor
    d. Los *métodos* getter y setter de los atributos
    e. El *método* __str__ , mostrando los valores de x e y del punto
    f. El *método* cuadrante() que devuelve un número entre 1 y 4 que indica el cuadrante en el cual se encuentra el Punto.



    g. El método distancia_al_centro() que devuelve un número que representa la distancia entre el punto y el centro de coordenadas. La fórmula a utilizar se muestra en la siguiente imagen:



    h. El *método* __eq__ , que compare los valores de 'x' e 'y' entre dos objetos de la clase Punto

5. Crear la clase **Segmento** que contiene:
    a. 2 atributos (punto_a, punto_b) del tipo **Punto** que representan los extremos del Segmento.
    b. El constructor de la clase **Segmento** que recibe dos parámetros del tipo **Punto** para los extremos

c. Los *métodos* getter y setter de los atributos
d. El *método* __str__ que retorne cómo está conformado el segmento (información de cada punto)
e. Un *método* longitud_segmento() que devuelve un número float que representa la longitud del segmento