Objetivos

1. Conocer y aplicar las funciones de los módulos `random` y `math` de Python. Aprender a buscar, estudiar y entender su documentación.
2. Enfrentarse a un problema complejo que no puede resolverse fácilmente con sólo una función.
3. Fomentar la habilidad de descomponer un problema en subproblemas y de implementar funciones que los resuelven, lo que se conoce comúnmente como la técnica de "Dividir y Conquistar".
4. Practicar el uso de instrucciones condicionales para la solución de problemas por casos.

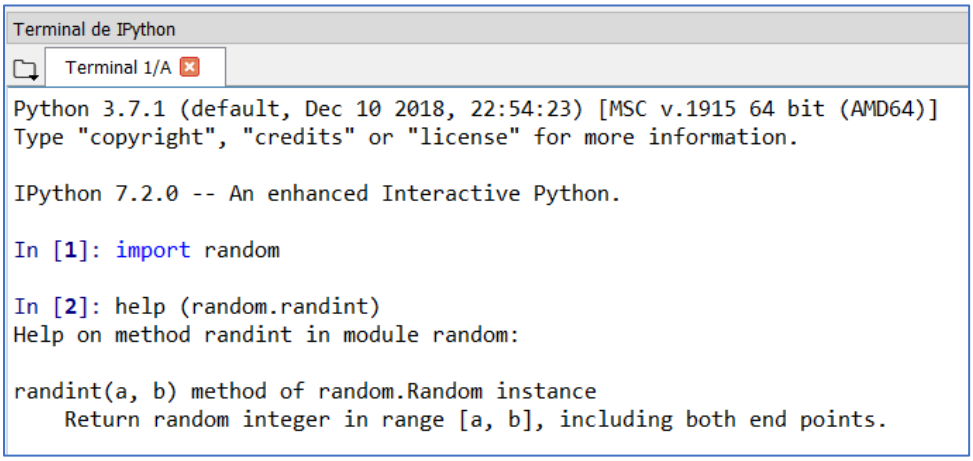
Contexto

En este laboratorio se van a hacer dos ejercicios relacionados con una feria o carnaval. Para cada ejercicio se describe el juego y las funciones que deben ser implementadas. En los ejercicios se sugieren funciones adicionales que deberían definirse para facilitar la implementación de las funciones principales.

Módulo math y random de Python

En este laboratorio se utilizarán dos módulos de Python: `math` y `random`, que implementan funciones que son necesarias para la solución de los ejercicios. La documentación de estos módulos está disponible en la página oficial de Python y se espera que usted sea capaz de buscarla, estudiarla y entenderla.

Recuerde que puede usar la función `help` en la terminal de Spyder para consultar la documentación de una función.



Módulo random

El módulo `random` contiene funciones para la generación de números aleatorios.
En este laboratorio se deben utilizar principalmente dos funciones de este módulo:

- `random()`

- `randint(a, b)`

La documentación completa se puede consultar en: https://docs.python.org/3/library/random.html

Módulo math

El módulo `math` contiene varias funciones matemáticas. Estas son algunas de ellas:

- `gcd(a,b)`
- `log(x, y), log2(x)`
- `sqrt(x)`
- `sin(x), cos(x), tan(x)`
- `degrees(x), radians(x)`

Este módulo también incluye algunas variables con valores especiales:

- `pi`
- `e`
- `inf`
- `nan`

La documentación completa se puede consultar en: https://docs.python.org/3/library/math.html

Actividad 0: Preparación

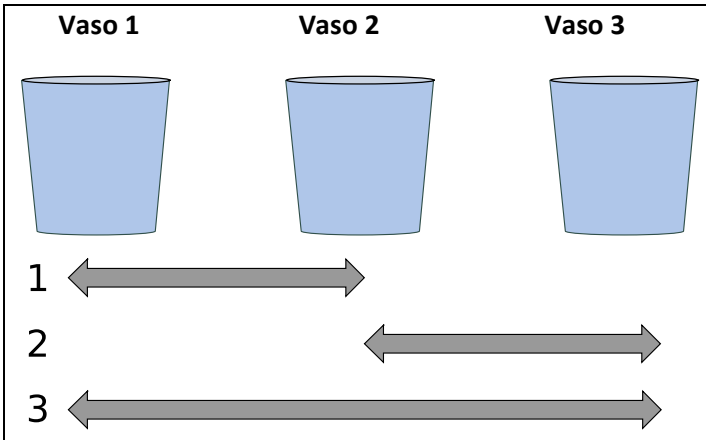
Cree una carpeta de trabajo y descargue allí el archivo N2-L2-Feria-esqueleto.zip que se encuentra adjunto a este enunciado en sicuaplus. Descomprima el archivo. Encontrará dos archivos: `consola_feria.py` y `funciones_feria.py`. El primero de estos corresponde a la interfaz de usuario que se le entrega completamente implementada y el segundo al módulo con el esqueleto de las funciones que usted debe completar en este laboratorio. Revise todas las funciones y lea con cuidado la documentación. Los lugares marcados con el comentario TODO indican los sitios donde usted debe realizar alguna modificación.

LEA COMPLETAMENTE Y CON CUIDADO CADA ACTIVIDAD ANTES DE EMPEZAR A RESOLVERLA

Actividad 1: Buscar la bolita

El juego de buscar una bolita escondida debajo de tres vasos es increíblemente popular no solo en ferias sino también en las calles de muchas ciudades. En este caso el juego se va a hacer de forma justa, sin ningún tipo de trampa: el jugador escoge en qué vaso se debe ubicar la bolita inicialmente, luego escoge en qué vaso va a terminar la bolita y finalmente la persona que maneja el juego hace tres movimientos aleatorios en los que intercambia dos vasos. Si la bolita termina en el vaso que escogió el jugador inicialmente, entonces es un ganador.

Hay 3 tipos posibles de intercambios que puede hacer la persona. Esos 3 tipos están representados en la siguiente figura:



Tipos de intercambios

Intercambio 1: Se intercambian los contenidos de los vasos 1 y 2.

Intercambio 2: Se intercambian los contenidos de los vasos 2 y 3.

Intercambio 3: Se intercambian los contenidos de los vasos 1 y 3.

Usted debe implementar una función llamada `buscar_bolita` que:

(1) recibe por parámetro la posición inicial de la bolita (vaso 1, 2 o 3) y la posición final a la que el jugador le apuesta (vaso 1, 2 o 3)
(2) selecciona aleatoriamente tres intercambios para que la persona realice sobre los vasos
(3) finalmente informa si el jugador ganó (`True`) o perdió (`False`)

Por ejemplo, el jugador podría indicar que la posición inicial de la bolita es el vaso 1 y que él piensa que terminará en el vaso 2. El azar podría luego determinar que los intercambios a realizar son de los tipos 1, 1 y 3, así que la bolita terminaría en el vaso 3 (pasa del vaso 1 al vaso 2, del vaso 2 al vaso 1 y del vaso 1 al vaso 3). Finalmente, su función informaría que el jugador perdió porque le apostó al vaso equivocado.

Vamos a descomponer el problema en varios subproblemas:

1. Preguntar al jugador (usuario) los datos de entrada: en qué vaso se debe ubicar la bolita inicialmente y en qué vaso cree que va a terminar la bolita.
2. Simular los 3 movimientos aleatorios para cambiar la bolita de un vaso a otro de acuerdo con los tipos de intercambio presentados en el diagrama.
3. Determinar si el jugador ganó dada su apuesta (el vaso en el que dijo que terminaría la bolita) y el contenido final de los vasos.
4. Informar el resultado final al usuario.

Ahora pensemos, de qué forma podemos resolver estos subproblemas:

1. Solicitar los datos de entrada al usuario es responsabilidad de la interfaz de usuario y se resuelve implementando una interfaz basada en consola.
2. Simular los 3 movimientos aleatorios es un problema que se puede resolver dentro de una función de nuestro módulo. Un movimiento aleatorio puede simularse utilizando la función `randint` del módulo `random` de Python para generar un número entero aleatorio entre 1 y 3 (para simular cada uno de los tipos de intercambio). Recuerde que es necesario hacer 3 movimientos aleatorios. A esta función la llamaremos `buscar_bolita`.
3. Para determinar si el jugador ganó o no, podemos igualmente implementar una función. A esta función la llamaremos `es_ganador`.
4. Informar el resultado final al usuario es también responsabilidad de la interfaz y lo haremos en la consola.

En la implementación que proponemos a continuación, la función `buscar_bolita` realiza varios pasos:

1. Simula los 3 movimientos aleatorios. Para representar el contenido de los vasos, sugerimos utilizar 3 variables booleanas dentro de la función (por ejemplo, vaso1, vaso2 y vaso3), que indiquen si en vaso 1, 2 o 3 está o no la bolita (True o False).
2. Llama a la función `es_ganador` para determinar si el jugador ganó o no.
3. Retorna la respuesta entregada por la función `es_ganador`.

A continuación se especifica la documentación de cada una de estas funciones.

```
def buscar_bolita(posicion_inicial: int, posicion_apuesta: int)->bool:
    """ Esta función ubica la bolita en la posición inicial indicada, hace 3 intercambios aleatorios y
    finalmente informa si la bolita quedó en la posición por la que apostó el jugador.
    Parámetros:
        posicion_inicial (int): es el número del vaso en el que va a quedar la bolita inicialmente (1, 2 o 3)
        posicion_apuesta (int): es el número del vaso en el que el jugador supone que va a quedar la bolita
        (1, 2 o 3)
    Retorno:
        Retorna True si la bolita quedó en la posición apostada y False en caso contrario.
    """

def es_ganador(posicion_apuesta, vaso1: bool, vaso2: bool, vaso3: bool)->bool:
    """ Esta función indica si el jugador ganó dependiendo de su apuesta y dependiendo del contenido de los
    vasos.
    Parámetros:
        posicion_apuesta (int): es la posición en la que el jugador supone que va a estar la bolita (1, 2 o
        3)
        vaso1 (bool): es True si la bolita se encuentra en el vaso 1 y False de lo contrario.
        vaso2 (bool): es True si la bolita se encuentra en el vaso 2 y False de lo contrario.
        vaso3 (bool): es True si la bolita se encuentra en el vaso 3 y False de lo contrario.
    Retorno:
        Retorna True si el jugador ganó su apuesta y False en caso contrario.
    """
```

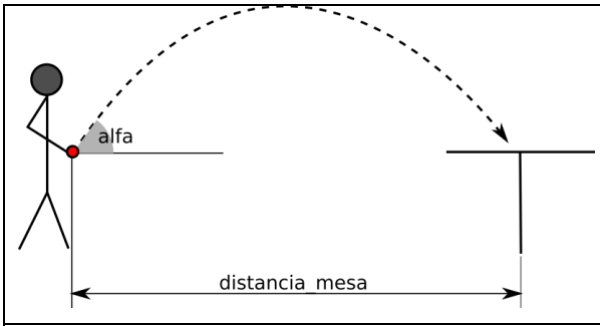
Actividad 2: Tiro al blanco

En este juego de tiro al blanco, el jugador tiene que lanzar una pequeña bola pegajosa a una mesa redonda de 40 cms de diámetro que se encuentra a 7 metros. La cantidad de puntos que recibe el jugador depende del lugar donde termine la bola pegajosa: 50 puntos en el círculo central (diámetro de 10 cms), 20 en el segundo círculo (diámetro de 20 cms), 10 puntos en el tercer círculo (diámetro de 30 cms) y 5 puntos en el cuarto círculo (diámetro de 40 cms).

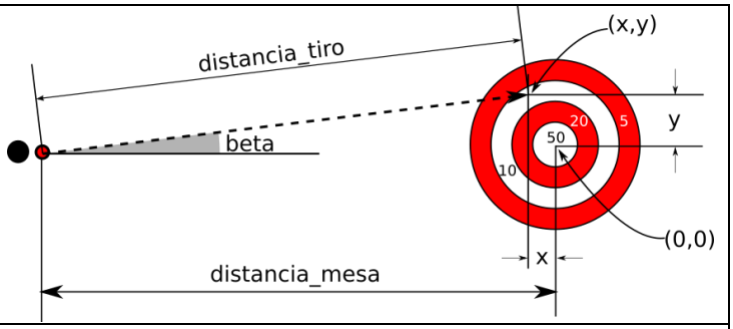
Este es un juego de habilidad, en el cual la bola se mueve describiendo un movimiento parabólico que se ve afectado por la gravedad de la Tierra. Cuando un jugador lanza la bola, lo que decide es la velocidad inicial del lanzamiento, el ángulo del lanzamiento con respecto a la horizontal (ángulo alfa) y el ángulo del lanzamiento con respecto a la línea que une la bola con el centro del blanco (ángulo beta).

Sin embargo, la mayoría de los jugadores tienen problemas con su puntería y su pulso que afectan sus lanzamientos. Por este motivo, el lanzamiento siempre tiene una incertidumbre:

- Velocidad inicial: += 10%
- Ángulo alfa: += 0.5º
- Ángulo beta: += 0.5º



a) Vista lateral



b) Vista superior

Usted debe implementar una función que recibe los datos de un lanzamiento (velocidad inicial, ángulo alfa y ángulo beta) e informa la cantidad de puntos obtenidos por el jugador.

Ayuda:

Puede suponer que la altura sobre el nivel del piso a la que parte la bola es la misma altura de la mesa. También puede suponer que la resistencia del aire es despreciable.

Recuerde que, dado que se desprecia la resistencia del aire, el movimiento parabólico puede descomponerse en un componente horizontal y un componente vertical. Para el componente horizontal se puede utilizar la siguiente ecuación básica del movimiento:

$$x = x_0 + v_x t$$

En este caso:

- *x* es la distancia en línea recta entre la posición inicial de la bola y la posición donde aterrizó (que calculará la función `calcular_distancia_tiro`).
- x_0 es la posición inicial del lanzamiento, que en este caso es 0.
- v_x es el componente horizontal de la velocidad inicial (que se calcula a partir de la velocidad inicial dada por el usuario).
- *t* es el tiempo que dura la bola en el aire (que calculará la función `calcular_tiempo_tiro`).

Los componentes horizontales y verticales de la velocidad pueden calcularse usando las siguientes fórmulas:

$$v_x = v_0 \cos \alpha$$
$$v_y = v_0 \sin \alpha$$

Donde:

- v_0 es la velocidad inicial del lanzamiento
- v_x es la velocidad inicial horizontal
- v_y es la velocidad inicial vertical

El tiempo que tarda la bola en el aire puede calcularlo con la siguiente ecuación (*G* es la gravedad y vale 9.81):

$$t = v_y / G$$

Finalmente, recuerde las siguientes definiciones de las funciones trigonométricas para triángulos rectángulos, que le serán útiles para calcular las coordenadas (x, y) del lugar donde cayó la bola a partir de la distancia horizontal recorrida por la bola:

$$\sin A = \frac{cateto\ opuesto}{hipotenusa} \qquad \cos A = \frac{cateto\ adyacente}{hipotenusa}$$

Vamos a descomponer el problema en 6 grandes subproblemas:

1. Preguntar al jugador (usuario) los datos de entrada: velocidad inicial, ángulo alfa y ángulo beta.
2. Simular los errores de puntería que afectan la velocidad inicial y los ángulos alfa y beta.
3. Calcular la distancia horizontal recorrida por la bola desde el lanzamiento hasta que caiga sobre la mesa.
4. Calcula la posición donde cayó la bola.
5. Calcular la cantidad de puntos que ganó el usuario dada la posición donde cayó la bola.
6. Informar el resultado final al usuario.

Ahora pensemos, de qué forma podemos resolver estos subproblemas:

1. Solicitar los datos de entrada al usuario es responsabilidad de la interfaz de usuario y se resuelve fácilmente implementando una interfaz basada en consola.
2. Simular los errores de puntería es un problema que se puede resolver dentro de una función de nuestro módulo. Un error aleatorio pude simularse utilizando la función `random` del módulo `random` de Python para generar un número flotante aleatorio entre 0 y 1. Esta simulación la haremos dentro de la función principal llamada `tiro_al_blanco`.
3. Para calcular la distancia horizontal recorrida por la bola desde el lanzamiento hasta que caiga sobre la mesa, podemos igualmente implementar una función. A esta función la llamaremos `calcular_distancia_tiro`. Note que esta función necesita calcular las velocidades horizontal y vertical iniciales y el tiempo que le toma a la bola caer sobre la mesa para poder calcular la distancia horizontal.
4. Para calcular el tiempo que le toma a la bola caer sobre la mesa, implementaremos la función `calcular_tiempo_tiro`.
5. El cálculo de la posición donde cayó la bola, lo dividiremos en dos funciones: `calcular_x_tiro` y `calcular_y_tiro` para calcular respectivamente las coordenadas (x, y) del lugar donde cayó la bola.
6. Para calcular la cantidad de puntos que ganó el usuario dada la posición donde cayó la bola, implementaremos la función `calcular_puntos`.
7. Por último, informar el resultado final al usuario es también responsabilidad de la interfaz y lo haremos en la consola.

Implemente las siguientes funciones. La función `tiro_al_blanco` es la función principal, mientras que las siguientes son funciones que le ayudarán a simplificar la función principal.

```
def tiro_al_blanco(velocidad_inicial: float, angulo_alfa: float, angulo_beta: float)->int:
    """ Esta función calcula los puntos obtenidos por un jugador dados los datos de su lanzamiento.
    La mesa se encuentra ubicada a 7 metros del lugar de lanzamiento.
    El blanco tiene 4 zonas de 10, 20, 30 y 40 cms de diámetro, que dan 50, 20, 10 y 5 puntos
    respectivamente.
    Parámetros:
        velocidad_inicial (float): la velocidad inicial en kilómetros/hora a la que sale la bola lanzada.
        angulo_alfa (float): el ángulo en grados al que sale la bola con respecto a la horizontal.
        angulo_beta (float): el ángulo en grados al que sale la bola con respecto a la línea recta entre el
        punto de lanzamiento y el centro del blanco.
    Retorno:
        La cantidad de puntos obtenidos por el jugador
    """
def calcular_distancia_tiro(velocidad_inicial: float, angulo_alfa: float, angulo_beta: float)->float:
    """ Calcula la distancia horizontal recorrida por la bola desde el lanzamiento hasta que caiga sobre la
    mesa.
    Parámetros:
        velocidad_inicial (float): la velocidad inicial en metros/segundo a la que sale la bola lanzada.
        angulo_alfa (float): el ángulo en grados al que sale la bola con respecto a la horizontal.
```

```
            angulo_beta (float): el ángulo en grados al que sale la bola con respecto a la línea recta entre el
            punto de lanzamiento y el centro del blanco.
        Retorno:
            La distancia en metros del lanzamiento
        """
def calcular_tiempo_tiro(velocidad_vertical_inicial: float)->float:
        """ Calcula el tiempo que le toma a la bola caer sobre la mesa.
        Parámetros:
            velocidad_vertical_inicial (float): la componente vertical de la velocidad inicial en
            metros/segundo a la que sale la bola lanzada.
        Retorno:
            La cantidad de segundos que le toma a la bola llegar a la mesa
        """

def calcular_x_tiro(distancia_tiro: float, distancia_mesa: int, angulo_beta: float)->float:
        """ Calcula la coordenada X del lugar donde cayó la bola, teniendo el centro de la mesa como la
        coordenada 0,0.
        Parámetros:
            distancia_tiro (float): la distancia en metros recorrida por la bola durante el lanzamiento
            distancia_mesa (float): la distancia en metros entre el lugar del lanzamiento y el centro de la
            mesa
            angulo_beta (float): el ángulo en grados al que sale la bola con respecto a la línea recta entre el
            punto de lanzamiento y el centro del blanco.
        Retorno:
            La coordenada x del lanzamiento, en centímetros
        """

def calcular_y_tiro(distancia_tiro: float, distancia_mesa: int, angulo_beta: float)->float:
        """ Calcula la coordenada Y del lugar donde cayó la bola, teniendo el centro de la mesa como la
        coordenada 0,0.
        Parámetros:
            distancia_tiro (float): la distancia en metros recorrida por la bola durante el lanzamiento
            distancia_mesa (float): la distancia en metros entre el lugar del lanzamiento y el centro de la
            mesa
            angulo_beta (float): el ángulo en grados al que sale la bola con respecto a la línea recta entre el
            punto de lanzamiento y el centro del blanco.
        Retorno:
            La coordenada y del lanzamiento, en centímetros
        """

def calcular_puntos(x: float, y: float, radio1: int, radio2: int, radio3: int, radio4: int)->int:
        """ Calcula la cantidad de puntos que ganó el usuario dada la posición donde cayó la bola.
        La coordenada 0,0 corresponde al centro de la mesa.
        Parametros:
            x (float): la coordenada x del lanzamiento, en centímetros
            y (float): la coordenada y del lanzamiento, en centímetros
            radio1 (int): el radio en centímetros de la zona 1 de la mesa (la zona más interna) que da 50
            puntos
            radio2 (int): el radio en centímetros de la zona 2 de la mesa que da 20 puntos
            radio3 (int): el radio en centímetros de la zona 3 de la mesa que da 10 puntos
            radio4 (int): el radio en centímetros de la zona 4 de la mesa (la zona más externa) que da 5 puntos
        Retorno:
            La cantidad de puntos obtenidos por el jugador
        """
```

Entrega

Debe entregar un solo archivo comprimido .zip con dos archivos:

- El módulo que contiene las funciones y
- La interfaz de usuario basada en consola que contiene el programa principal

El nombre del archivo comprimido tiene que seguir la siguiente estructura:
n2-l2-login.zip

donde login corresponde a su nombre de usuario de Uniandes. Por ejemplo, si su usuario es p.perez10, su archivo debe llamarse: n2-l2-p.perez10.zip. Entregue el archivo a través de Brightspace en la tarea designada como "**N2-L2**".