Programación & Decisiones

Semestre 2022–2

Taller 2

Instructor: Daniel Aguirre

Fecha de entrega máxima: Viernes 23 de septiembre 11:59pm

Reglas:

Debe enviar un archivo comprimido (.zip) con la estructura: T2_G<# del grupo>.zip (ej: T2_G3.zip)

La parte 1 debe ir en un archivo que se llame retos.py y solo deben estar las definiciones de funciones y clases. La parte 2 es una carpeta que se especifica en su sección correspondiente.

Por ejemplo, la estructura del grupo 1 se debe ver así

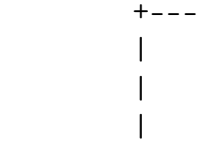
```
+T2_G1
 |__retos.py
 |__ahorcado
    |__logica.py
    |__interfaz.py
```

Parte I: Retos

Reto 1: Tiro al blanco

Cree una función que reciba una matriz(i.e., lista de listas) de tamaño nxm que contiene una "x"(objetivo) y ">" como una flecha hacia esa dirección. La función debe retornar True si la flecha va dirigida hacia el blanco. Retorna False si la flecha no va en dirección al blanco.

Representación gráfica	En python	Resultado

	`[[" ", " ", " ", " "],`
	` [" ", ">", " ", "x"],`
	` [" ", " ", " ", " "]]`
		True

	`[[" ", ">", " ", " "],`
	` [" ", " ", " ", "x"],`
	` [" ", " ", " ", " "]]`
		False

	`[[" ", " ", " ", " "],`
	` [" ", "x", ">", " "],`
	` [" ", " ", " ", " "]]`
		False

Nombre de función: es_tiro_al_blanco

Input:
matrix: (list) lista de listas que contiene siempre un objetivo/blanco "x" y una flecha ">"

Output: (bool) retorna True si la flecha va en dirección al objetivo, False de lo contrario.

Pista: Puede realizar un recorrido directo por los elementos (no por posiciones) y crear una flag para saber si la flecha está presente en una dirección!.

Reto 2: Suma de diagonales

Dada una matriz nxn (matriz cuadrada), retorne la suma de las dos diagonales de la matriz.

Representación gráfica	En python	Resultado
+----------+ | 1 5 7 8 | | 2 3 4 1 | | 5 5 4 9 | | 6 7 2 6 | +----------+	[[1, 5, 7, 8], [2, 3, 4, 1], [5, 5, 4, 9], [6, 7, 2, 6]]	d1: 1+3+4+6=14 d2: 8+4+5+6=23 s = d1+d2=37
+---------+ | 1 4 6 | | 2 3 4 | | 6 1 2 | +---------+	[[1, 4, 6], [2, 3, 4], [6, 1, 2]]	d1: 1+3+2=6 d2: 6+3+6=15 s = d1+d2= 21
+----------+ | | | | | | +----------+	[[]]	0

Nombre de función: sumar_diagonales

Input:
matrix: (list) lista de listas que representa a una matriz cuadrada

Output: (int) La suma de las diagonales de una matriz cuadrada cualquiera

Pista: Traten de encontrar el patrón de las posiciones que se suman en cada una de las diagonales.

Reto 3: Construyendo esferas

Construya una clase **Esfera** que reciba por parámetros en el constructor el radio y su masa. Note que estos serán los atributos.
Atributos:

- **radio:** radio de la esfera creada
- **masa:** masa de la esfera creada

Métodos:

- get_radio() => retorna el radio de la esfera.
- get_masa() => retorna la masa de la esfera
- get_volumen() => retorna el volumen de la esfera (redondeado a 2 decimales)
- get_area_superficie() => retorna el area de superficie de la esfera.
- get_densidad() => retorna la densidad de la esfera.

Ejemplo:

```
esfera = Esfera(2,50)
esfera.get_radio() => 2
esfera.get_masa() => 50
esfera.get_volumen() => 33.51
esfera.get_area_superficie() => 50.26
esfera.get_densidad() => 1.49
```

Nota: Si no conoce como se obtiene el volumen, area y densidad de una esfera googleelos. **Pista:** Si requiere el número π para sus calculos, puede utilizarlo con el módulo de math. Investigue en internet como utilizarlo.

Reto 4: Quarks

Un físico está trabajando en un proyecto de investigación y quiere crear un modelo de interacción entre quarks. Sin embargo, no encuentra la forma correcta de hacer la representación en programación. Ustedes, que conocen la programación orientada a objetos (POO), ofrecen su ayuda para que el físico pueda crear multiples objetos quarks y pueda modelar estas interacciones.

Para esto, cree una clase **Quark** que tenga 3 atributos: **color**, **sabor** y **numero_baryon**. El objeto debe ser creado a través de un constructor que toma por parámetros el color y el sabor. El numero baryon es el mismo para todos los quarks: 1/3

Atributos:

- **color:** hay 3 tipos de colores de quarks: 'rojo', 'azul' y 'verde'

- **sabor:** hay 6 tipos de sabores: 'up', 'down', 'strange', 'charm', 'top', y 'bottom'.
- **numero_baryon:** 1/3 para cualquier quark.

Métodos:

- interactuar(quark) -> recibe como parámetro un objeto Quark. Cuando los quarks interactuan, intercambian sus colores. Es decir, el método debe intercambiar los colores de los dos quarks (el quark del que se llama el método y el quark que ingresa por parámetro).

Ejemplo:

```
q1 = Quark("rojo", "up")
q1.color => "rojo"
q1.sabor => "up"
q2 = Quark("azul", "strange")
q2.color => "azul"
q2.numero_baryon => 0.333...
q1.interactuar(q2)
q1.color => "azul"
q2.color => "rojo"
```

Parte II: El juego de ahorcado

Utilizaremos el patrón de modularización visto en clase para crear el juego de ahorcado. Para esto, cree una carpeta que se llame **ahorcado** y que contenga dos archivos: **logica.py** e **interfaz.py**. Se dará una guía para la creación de cada archivo. Es decir, no deben crear el juego desde 0. **SOLAMENTE** deben implementar las funciones (asegúrense que las funciones estén bien) y seguir las indicaciones en cada paso.

```
+ahorcado
  |__logica.py
  |__interfaz.py
```

Construcción del archivo de lógica

Para la construcción de este archivo, deberán crear todas las funciones core del juego similar a lo visto en clase. Serán 5 funciones bastante simples.

Nombre de función 1: crear_palabra_vacia
Input: palabra_secreta: (str) la palabra secreta que deberá adivinar el jugador
Output: (str) retorna una serie de asteriscos "*" del mismo tamaño de la palabra secreta. Esto simulará los guiones del ahorcado y la palabra actual del jugador
Ejemplo:
crear_palabra_vacia("vaso") => "****"

Nombre de función 2: es_palabra_correcta
Input:

palabra_secreta: (str) La palabra que el jugador debe adivinar
palabra_actual: (str) La palabra actual que el jugador tiene
Output: (bool) retorna True si es la misma palabra, False D.L.C
Ejemplos:
es_palabra_correcta("hola", "hola") => True
es_palabra_correcta("hola", "*ol*") => False

Nombre de función 3: obtener_palabra_random
Input: palabras: (list) lista de palabras que se utilizará en el juego
Output: (str) retorna una palabra random de la lista de palabras
Ejemplo:
obtener_palabra_random(["hola", "daniel", "computador"]) => puede retornar "hola" o "daniel" o "computador"
Nota: Puede importar el modulo random y utilizar la función choice. Investigue como utilizar esta función en internet.

Nombre de función 4: calcular_total_rondas
Para calcular el total de rondas, debe obtener el mínimo número de intentos que se necesitan para ganar el juego y luego multiplicar por una razón para obtener el total de rondas que se le dará al participante. Note que **no** siempre el mínimo número de rondas es el tamaño de la palabra secreta. Por ejemplo, el mínimo número de intentos para "hola" es 4, sin embargo, para "holo" es 3, ya que cuando el usuario intenta la "o" se desbloquean dos letras en la palabra y no solo 1. Las razones dependerán del nivel de dificultad. A continuación está la lista:

```
Dificultad | Razón
    1      |   2
    2      |  1.7
    3      |  1.5
    4      |  1.3
```

Si la palabra es "hola" y la dificultad es fácil (1), el mínimo número de intentos es 4 y la razón 2. Entonces el total de rondas que se le dará al participante para adivinar la palabra es 8 rondas. Si el total de rondas es un número decimal, debe redondearlo al siguiente entero (función ceil del módulo math)
Input:
palabra_secreta: (str) la palabra secreta que deberá adivinar el jugador
dificultad: (int) un número del 1 al 4 donde hace referencia al nivel de dificultad. Siendo 1 más fácil y 4 más difícil.
Output: (int) El total de rondas que tendrá el jugador para adivinar la palabra
Ejemplos:
calcular_total_rondas("vaso", 1) => 8
calcular_total_rondas("holo", 1) => 6
calcular_total_rondas("holo", 3) => 5

Nombre de función 5: obtener_nueva_palabra
El intento por lo general es una letra. Si el usuario escoge una palabra como intento, es decir, si el tamaño de la string de intento es mayor a 1, la función evalúa si el intento es igual a la palabra secreta, en caso de ser igual, retorna la palabra secreta. De lo contrario, retorna la palabra actual sin modificaciones. Si el usuario escoge una letra como intento (i.e., el tamaño de la string intento es 1) entonces evalúa si la letra está presenta en la palabra secreta. Si lo

está, retorna la palabra actual actualizada con la nueva letra. Si la letra no está en la palabra secreta, retorna la palabra actual sin modificaciones.
Input:
palabra_secreta: (str) la palabra secreta que deberá adivinar el jugador
palabra_actual: (str) La palabra actual que el jugador tiene
intento: (str) La letra o palabra que el jugador escoge
Output: (str) Si el intento es una letra, verifica que el intento esté en la palabra secreta, entonces completa la palabra actual con la nueva letra. De lo contrario, retorna la palabra actual sin modificar. Si el intento es una palabra, verifica que el intento sea igual a la palabra secreta. Si son iguales retorna la palabra secreta. De lo contrario, retorna la palabra actual sin modificaciones.
Ejemplos:
obtener_nueva_palabra("vaso", "****", "a") => "*a**"
obtener_nueva_palabra("hola", "**la", "h") => "h*la"
obtener_nueva_palabra("daniel", "da****", "x") => "da****"
obtener_nueva_palabra("coco", "****", "o") => "*o*o"
obtener_nueva_palabra("computador", "*******dor", "cargador") => "*******dor"
obtener_nueva_palabra("computador", "*******dor", "c") => "c******dor"

Construcción del archivo de interfaz

El archivo de interfaz contiene todo lo que el usuario visualizará y la ejecución de funciones del archivo de lógica. Recuerde importar el archivo de lógica para hacer uso de sus funciones. Puede basarse en el juego de picas y fijas y en la calculadora realizados en clase para crear este archivo. A continuación les daré una guía de implementación:

Nombre de función 1: ejecutar_crear_palabra_vacia
Esta función solo llamará y ejecutará la función crear_palabra_vacia del archivo de lógica.
Input: palabra_secreta: (str) la palabra secreta que deberá adivinar el jugador
Output: (str) retorna una serie de asteriscos "*" del mismo tamaño de la palabra secreta. Esto simulará los guiones del ahorcado y la palabra actual del jugador
Ejemplo:
ejecutar_crear_palabra_vacia("vaso") => "****"

Nombre de función 2: ejecutar_obtener_palabra_random
Esta función tendrá almacenado una lista de palabras creadas manualmente y deberá llamar a la función de obtener palabra random para retornar una de las palabras de la lista.
Input: Output: (str) retorna una palabra random de la lista de palabras
Ejemplo:
ejecutar_obtener_palabra_random() => "hola"
Nota: Puede basarse en el juego de picas y fijas para crear esta función.

Nombre de función 3: ejecutar_calcular_total_rondas
Esta función simplemente ejecuta la función de lógica calcular total rondas. Sin embargo, dado que es la interfaz de usuario, los parámetros que recibe son strings. Debe castear los parámetros correspondientes para que puedan llamarlos correctamente en su función de lógica.
Input:

palabra_secreta: (str) la palabra secreta que deberá adivinar el jugador
dificultad: (str) un número del 1 al 4 en formato de string. Para llamarlo en su función de lógica debe convertirlo a dato numérico
Output: (int) El total de rondas que tendrá el jugador para adivinar la palabra
Ejemplos:
calcular_total_rondas("vaso", 1) => 8
calcular_total_rondas("holo", 1) => 6
calcular_total_rondas("holo", 3) => 5

Nombre de función 4: ejecutar_obtener_nueva_palabra
Esta función simplemente ejecuta la función de obtener nueva palabra de su archivo de lógica
Input:
palabra_secreta: (str) la palabra secreta que deberá adivinar el jugador
palabra_actual: (str) La palabra actual que el jugador tiene
intento: (str) La letra o palabra que el jugador escoge
Output: (str) Si la letra(intento) está en la palabra secreta, entonces completa la palabra actual con la nueva letra. De lo contrario, retorna la palabra actual sin modificar
Ejemplo:
ejecutar_obtener_nueva_palabra("vaso", "****", "a") => "*a**"

Nombre de función 5: imprimir_menu
Imprime el siguiente menu:

```
BIENVENIDO AL JUEGO DE AHORCADO
-------------------------------
1. Cambiar el nivel de dificultad
2. Jugar
3. Salir
```

Nombre de función 6: imprimir_niveles_dificultad
Imprime el siguiente menu de niveles de dificultad:

```
NIVELES DE DIFICULTAD
---------------------
1. Fácil
2. Normal
3. Difícil
4. Locura
```

Nombre de la función 7: iniciar_juego
Esta función recibe por parámetro la dificultad del juego. No tendrán que implementarla, sin embargo, asegúrense de entender el código:

```
def iniciar_juego(dificultad: int):
    ronda = 1
    palabra_secreta = ejecutar_obtener_palabra_random()
    total_rondas = ejecutar_calcular_total_rondas(palabra_secreta, dificultad)
    palabra_actual = ejecutar_crear_palabra_vacia(palabra_secreta)
    ganador = False
    print("Tamaño de la palabra a adivinar:", palabra_actual)
    while ronda <= total_rondas:
```

```
            print("\n-------")
            print("RONDA", ronda)
            print("-------")
            intento = input("Escriba una letra o una palabra: ")
            palabra_actual = ejecutar_obtener_nueva_palabra(palabra_secreta,
                                                            palabra_actual, intento)
            print("palabra actualizada:", palabra_actual)
            if lg.es_palabra_correcta(palabra_secreta, palabra_actual):
                ganador = True
                break
            ronda += 1

        if ganador == True:
            print("FELICITACIONES, GANASTE EL JUEGO!")
        else:
            print("LO SIENTO, HAS PERDIDO :(")
            print("LA PALABRA ERA: ", palabra_secreta)
        print("")
```

NOTA: En el archivo de interfaz utilicé *import logica as lg*. Por esta razón se hace el llamado de es_palabra_correcta como lg.es_palabra_correcta(...). Sin embargo, si utilizó otro tipo de import, corrija esta función.

Nombre de la función 8: main
Esta función se encargará de iniciar la aplicación. De igual forma, no tendrán que implementarla, pero deben entender el código a la perfección.

```
    def main():
        dificultad = 1
        while True:
            imprimir_menu()
            opcion = input("Seleccione opción: ")
            if opcion == "1":
                imprimir_niveles_dificultad()
                dificultad = int(input("Seleccione nivel de dificultad: "))
            elif opcion == "2":
                iniciar_juego(dificultad)
            elif opcion == "3":
                print("Gracias por jugar, vuelva pronto!")
                break
```