**EJERCICIO DE APLICACIÓN 02**

Este conjunto de problemas le dará a conocer el tema de la creación de funciones en Python, así como mecanismos de bucle para repetir un proceso de cálculo hasta que se alcanza una condición. Deberá guardar sus códigos (3) con su primer nombre y su primer apellido, seguido del problema de aplicación y la parte del problema que representa el código pa02\_parte1.py (Lois\_Lane\_pa02\_parte1.py, Lois\_Lane\_pa02\_parte2.py, Lois\_Lane\_pa02\_parte3.py). ¡No olvide incluir comentarios para ayudar a entender su código!

**Colaboración**

Puedes trabajar en parejas. Sin embargo, cada estudiante debe escribir y entregar su tarea por separado. Asegúrese de indicar con quién ha trabajado en los comentarios de su presentación.

**PROBLEMA 1: EL AHORCADO BÁSICO**

Vas a implementar una variación del juego clásico de palabras Ahorcado. Si no estás familiarizado con las reglas del juego, puedes leer <https://es.wikipedia.org/wiki/Ahorcado_(juego)>. ¡No te sientas intimidado por este problema, realmente es más fácil de lo que parece! Vamos a abordar este problema, guiándolo a través de la creación de funciones de ayuda antes de implementar el juego real.

1. **INTRODUCCIÓN**

Descarga los archivos “ahorcado\_detallado.py” y “palabras.txt” y **guarda ambos en el mismo directorio**. Ejecuta el archivo ahorcado.py antes de escribir algún código para asegurarse que sus archivos se guardan correctamente. El código que se te ha dado carga las palabras desde un archivo. Deberías ver la siguiente salida en el Shell de IPython:

*Cargando lista de palabras desde archivo...*

*78566 palabras cargadas.*

**Si ve el texto anterior, continúe con los Requisitos del Juego Ahorcado**.

¡Si no lo hace, vuelva a verificar que ambos archivos estén guardados en el mismo lugar!

1. **REQUISITOS DEL JUEGO AHORCADOS**

Implementarás una función llamada *hangman* que permitirá al usuario jugar al ahorcado contra la computadora. La computadora elige la palabra y el jugador intenta adivinar las letras de la palabra.

Aquí está el comportamiento general que queremos implementar. ¡No te intimides! Esta es solo una descripción: **dividiremos esto en pasos y proporcionaremos más especificaciones funcionales más adelante en la descripción del problema, así que ¡sigue leyendo!**

1. La computadora debe seleccionar una palabra al azar de la lista de palabras disponibles que se proporcionó en words.txt

**Tenga en cuenta que palabras.txt contiene palabras en letras minúsculas.**

1. El usuario recibe un cierto número de conjeturas al comienzo.
2. El juego es interactivo, el usuario ingresa su conjetura y la computadora:
3. revela la letra si existe en la palabra secreta
4. penalizar al usuario y actualiza el número de conjeturas restantes
5. El juego termina cuando el usuario adivina la palabra secreta o el usuario se queda sin intentos.

**PROBLEMA 2.**

**AHORCADO PARTE 1: TRES FUNCIONES DE AYUDA**

Antes de que tengas que escribir el código para organizar el juego del ahorcado, vamos a dividir el problema en subtareas lógicas, creando tres funciones auxiliares que necesitarás tener para que este juego funcione. Este es un enfoque común para la resolución de problemas computacionales, y uno que queremos que comiences a experimentar.

El archivo ahorcado.py tiene una serie de funciones ya implementadas que puede utilizar al redactar su solución. Puede ignorar el código en las dos funciones en la parte superior del archivo que ya se han implementado para usted, aunque debe comprender cómo usar cada función auxiliar leyendo los docstrings.

**1A) Determine si la palabra ha sido adivinada**

Primero, implemente la función *is\_word\_guessed* que toma dos parámetros, una cadena, *secret\_word* y una lista de letras (strings), *letters\_guessed*. Esta función devuelve un booleano *True* si *secret\_word* ha sido adivinado (es decir, todas las letras de *secret\_word* están *letters\_guessed*), y *False* en caso contrario. Esta función será útil para ayudarlo a decidir cuándo se ha completado con éxito el juego del ahorcado, y se convierte en una prueba final para cualquier ciclo iterativo que verifique las letras con la palabra secreta.

Para esta función, puede suponer que todas las letras en secret\_word y letters\_guessed están en minúsculas.

**Ejemplo de Uso:**

*>>> secret\_word = 'manzana'*

*>>> letters\_guessed = ['a', 'i', 'm', 'p', 'r', 's']*

*>>> print(is\_word\_guessed(secret\_word, letters\_guessed))*

*False*

**1B) Obtener los intentos del usuario**

A continuación, implemente la función *get\_guessed\_word* que toma dos parámetros, una cadena, *secret\_word* y una lista de letras, *letters\_guessed*. Esta función devuelve una cadena compuesta por letras y guiones bajos, en función de las letras de *letters\_guessed* en *secret\_word*. ¡Esto no debería ser muy diferente de *is\_word\_guessed*!

Vamos a usar un guion bajo seguido de un espacio (\_) para representar letras desconocidas. Podríamos haber elegido otros símbolos, pero la combinación de guion bajo y espacio es visible y fácil de discernir. Tenga en cuenta que el espacio es muy importante, ya que de lo contrario es difícil distinguir si \_\_\_\_ tiene cuatro elementos de longitud o tres. Esto se llama usabilidad, es muy importante, cuando se programa, considerar la usabilidad de su programa. Si los usuarios consideran que su programa es difícil de entender u operar, ¡no lo usarán! Le recomendamos que piense en la usabilidad cuando diseñe su programa.

**Consejo**: al diseñar su función, piense qué información desea devolver. cuando haya terminado, si necesita un lugar para almacenar esa información mientras itera sobre una estructura de datos y cómo desea agregar información a su resultado acumulado.

**Ejemplo de Uso:**

*>>> secret\_word = 'manzana'*

*>>> letters\_guessed = ['a', 'i', 'm', 'p', 'r', 's']*

*>>> print(get\_guess\_word(secret\_word, letters\_guessed))*

*'ma\_ \_ a\_ a'*

**1C) Obtener todas las letras disponibles**

Ahora, implemente la función *get\_available\_letters* que toma un parámetro - una lista de letras, *letters\_guessed*. Esta función devuelve una cadena compuesta de letras minúsculas en español - todas letras minúsculas en español que no están en *letters\_guessed*.

Esta función debe devolver las letras en orden alfabético. Para esta función, puede suponer que todas las letras en *letters\_guessed* son minúsculas.

**Consejo**: puede considerar el uso de *string.ascii\_lowercase*, que es una cadena compuesta de todas las letras minúsculas:

*>>> import string*

*>>> print(string.ascii\_lowercase)*

*abcdefghijklmnopqrstuvwxyz*

**Ejemplo de Uso:**

*>>> letters\_guessed = ['a', 'i', 'm', 'p', 'r', 's']*

*>>> print(get\_available\_letters(letters\_guessed))*

bcdefghjklnoqtuvwxyz

**PROBLEMA 3**

**EL AHORCADO PARTE 2: EL JUEGO**

Ahora que ha creado algunas funciones útiles, puede pasar a implementar la función *hangman*, que toma un parámetro - *secret\_word* que el usuario debe adivinar. Inicialmente, puede (¡y debería!) configurar manualmente esta palabra secreta cuando ejecute esta función - esto hará que sea más fácil probar su código. Pero al final, deseará que la computadora seleccione esta palabra secreta al azar antes de invitarlo a usted u otro usuario a jugar el juego ejecutando esta función.

Llamar a la función *hangman* inicia un juego interactivo de Ahorcado entre el usuario y la computadora. Al diseñar su código, asegúrese de aprovechar las tres funciones de ayuda, *is\_word\_guessed*, *get\_guessed\_word* y *get\_available\_letters*, que ha definido en la parte anterior.

A continuación se detallan los requisitos del juego divididos en diferentes categorías. ¡Asegúrese de que su implementación cumpla con todos los requisitos!

**Requisitos del Juego**

1. **Arquitectura del Juego**
2. La computadora debe seleccionar una palabra al azar de la lista de palabras disponibles que se proporcionó en palabras.txt. Las funciones para cargar la lista de palabras y seleccionar una palabra aleatoria ya se han proporcionado en ahorcado.py.
3. Los usuarios comienzan con 6 intentos.
4. Al comienzo del juego, hazle saber al usuario cuántas letras contiene la palabra de la computadora y con cuántos intentos comienza.
5. La computadora realiza un seguimiento de todas las letras que el usuario no ha adivinado hasta el momento y antes de cada turno muestra al usuario las "letras restantes"

**Ejemplo de implementación del juego**

*Cargando lista de palabras desde archivo...*

*78566 palabras cargadas.*

*Bienvenido al juego de Ahorcados!*

*Estoy pensando en una palabra que tiene 4 letras*

*---------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: abcdefghijklmnopqrstuvwxyz*

1. **Interacción Usuario-Computadora**

El juego debe ser interactivo y fluir de la siguiente manera:

1. Antes de cada intento, debe mostrar al usuario:
2. Recuérdele al usuario cuántos intentos le restan después de cada intento.
3. todas las letras que el usuario aún no ha adivinado
4. Pídale al usuario que proporcione un intento a la vez. (Consulte los requisitos de entrada del usuario a continuación para ver qué tipos de entradas puede esperar del usuario)
5. Inmediatamente después de cada intento, al usuario se le debe decir si la letra está en la palabra de la computadora.
6. Después de cada intento, también debe mostrar al usuario la palabra de la computadora, con letras adivinadas y letras sin repetir reemplazadas con un guion bajo y espacio (\_ )
7. Al final del intento, imprima algunos guiones (------) para ayudar a separar las suposiciones individuales entre sí

**Ejemplo de implementación del juego**

(El color azul que se muestra a continuación solo está allí para mostrarle lo que el usuario ingresó, a diferencia de lo que arrojó la computadora).

*Tienes 6 intentos restantes*

*Letras disponibles: abcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: a*

*Buen intento: \_ a\_ \_*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: b*

*Oops! Esa letra no está en mi palabra: \_ a\_ \_*

1. **Requisitos de entrada del usuario**
2. Puede suponer que el usuario solo adivinará un carácter a la vez, pero el usuario puede elegir cualquier número, símbolo o letra. ¡Su código debería aceptar letras mayúsculas y minúsculas como intentos válidos!
3. Si el usuario ingresa algo además de una letra del alfabeto (símbolos, números), dígale al usuario que solo pueden ingresar una letra del alfabeto. Debido a que el usuario puede hacer esto por accidente, deberían recibir 3 advertencias al comienzo del juego. Cada vez que ingresan una entrada no válida o una letra que ya han adivinado, deben perder una advertencia. Si el usuario no tiene advertencias e ingresa una entrada no válida, debería perder un intento.

**Consejo #1**: Use llamadas a la función *input* para que el usuario ingrese el intento.

1. Verifique que la entrada del usuario sea un alfabeto
2. Si el usuario no ingresa letras mayúsculas o minúsculas, reste una advertencia o un intento.

**Consejo #2**: es posible que las funciones de cadena *str.isalpha ('su cadena')* y *str.lower ('Su cadena')* sean útiles. Si no sabe cuáles son estas funciones, puede intentar escribir *help (str.isalpha)* o *help (str.lower)* en su shell Spyder para ver la documentación de las funciones.

**Consejo #3**: dado que las palabras en palabras.txt son minúsculas, es posible que sea más fácil convertir la entrada del usuario a minúsculas en todo momento y que su juego solo se maneje en minúscula.

**Ejemplo de implementación del juego**

*Tienes 3 advertencias restantes*

*Tienes 6 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: s*

*Oops! Esa letra no está en mi palabra: \_ a\_ \_*

*----------------------*

*Tienes 5 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrtuvwxyz*

*Por favor ingresa una letra: $*

*Oops! Esa no es una letra válida. Te quedan 2 advertencias: \_ a\_ \_*

1. **Reglas del juego**
2. El usuario comienza con 3 advertencias.
3. Si el usuario ingresa algo además de una letra del alfabeto (símbolos, números), dígale al usuario que solo pueden ingresar una letra del alfabeto.
4. Si el usuario tiene una o más advertencias, el usuario debería perder una advertencia. Indique al usuario el número de advertencias restantes.
5. Si el usuario no tiene advertencias restantes, debería perder un intento.
6. Si el usuario ingresa una letra que ya ha sido adivinada, imprima un mensaje diciéndole al usuario que la letra ya ha sido adivinada anteriormente.
7. Si el usuario tiene una o más advertencias, el usuario debería perder una advertencia. Indique al usuario el número de advertencias restantes.
8. Si el usuario no tiene advertencias, deberían perder un intento.
9. Si el usuario ingresa una letra que no se ha adivinado antes y la letra está en la palabra secreta, el usuario no pierde ningún intento.
10. **Consonantes**: si el usuario ingresa una consonante que no ha sido adivinada y la consonante no está en la palabra secreta, el usuario pierde **un** intento si es una consonante.
11. **Vocales**: si la vocal no ha sido adivinada y la vocal no está en la palabra secreta, el usuario pierde **dos** intentos. Las vocales son a, e, i, o y u.

**Ejemplo de implementación del juego**

*Tienes 5 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrtuvwxyz*

*Por favor ingresa una letra: m*

*Buen intento: ma\_ \_*

*----------------------*

*Tienes 5 intentos restantes*

*Letras disponibles: bcdefghijklnopqrtuvwxyz*

*Por favor ingresa una letra: e*

*Oops! Esa letra no está en mi palabra: ma\_ \_*

*----------------------*

*Tienes 3 intentos restantes*

*Letras disponibles: bcdfghijklnopqrtuvwxyz*

*Por favor ingresa una letra: e*

*Oops! Ya has intentado esa letra. Te quedan 2 advertencias: ma\_ \_*

1. **Fin del juego**
2. El juego debe finalizar cuando el usuario construye la palabra completa o se queda sin intentos.
3. Si el jugador se queda sin intentos antes de completar la palabra, dígales que perdieron y revele la palabra al usuario cuando termine el juego.
4. Si el usuario gana, imprima un mensaje de felicitación y dígale al usuario su puntaje.
5. El puntaje total es el número de *guesses\_remaining* una vez que el usuario ha adivinado la *secret\_word* por el número de letras únicas en *secret\_word*.

**Total score = guesses\_remaining\* número de letras únicas en secret\_word**

**Ejemplo de implementación del juego**

*Tienes 3 intentos restantes*

*Letras disponibles: bcdefghijklnopqrtuvwxyz*

*Por favor ingresa una letra: n*

*Buen intento: man\_*

*----------------------*

*Tienes 3 intentos restantes*

*Letras disponibles: bcdefghijklmopqrtuvwxyz*

*Por favor ingresa una letra: o*

*Buen intento: mano*

*----------------------*

*¡Felicitaciones, has ganado!*

*Tu puntaje total para este juego es: 12*

**Ejemplo de implementación del juego**

*Tienes 3 intentos restantes*

*Letras disponibles: bcdefghijklnopqrtuvwxyz*

*Por favor ingresa una letra: n*

*Buen intento: delfin*

*----------------------*

*¡Felicitaciones, has ganado!*

*Tu puntaje total para este juego es: 18*

1. **Sugerencias generales**
2. Considere escribir funciones auxiliares adicionales si las necesita.
3. Hay cuatro piezas importantes de información que puede desear almacenar:
4. *secret\_word*: La palabra a adivinar. Esto ya se usa como el nombre del parámetro para la función *hangman*.
5. *letters\_guessed*: Las letras que se han adivinado hasta ahora. Si adivinan una letra que ya está en *letters\_guessed*, debe imprimir un mensaje indicándoles que ya lo adivinaron, pero no penalizarlo por ello.
6. *guesses\_remaining*: el número de intentos que el usuario tiene. Tenga en cuenta que en nuestro juego de ejemplo, la penalidad por elegir una vocal incorrecta es diferente de la penalización por elegir una consonante incorrecta.
7. *warnings\_remaining*: la cantidad de advertencias que le queda al usuario. Tenga en cuenta que un usuario solo pierde una advertencia al ingresar un símbolo o una letra que ya ha sido adivinada.
8. **Ejemplo de juego**

Mire cuidadosamente los ejemplos dados arriba para la ejecución del ahorcado, ya que eso sugiere ejemplos de información que querrá imprimir después de cada adivinanza de una letra.

**Nota: ¡Intente hacer sus declaraciones de impresión lo más cerca posible del juego de ejemplo!**

La salida de un juego **ganador** debería verse así. (El color azul que se muestra a continuación solo está allí para mostrarle lo que el usuario ingresó, a diferencia de lo que arrojó la computadora).

*Cargando lista de palabras desde archivo...*

*78566 palabras cargadas.*

*Bienvenido al juego de Ahorcados!*

*Estoy pensando en una palabra que tiene 4 letras*

*Tienes 3 advertencias restantes*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: abcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: a*

*Buen intento: \_ a\_ \_*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: a*

*Oops! Ya has intentado esa letra. Te quedan 2 advertencias: \_ a\_ \_*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: s*

*Oops! Esa letra no está en mi palabra: \_ a\_ \_*

*----------------------*

*Tienes 5 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrtuvwxyz*

*Por favor ingresa una letra: $*

*Oops! Esa no es una letra válida. Te quedan 1 advertencias: \_ a\_ \_*

*----------------------*

*Tienes 5 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrtuvwxyz*

*Por favor ingresa una letra: m*

*Buen intento: ma\_ \_*

*----------------------*

*Tienes 5 intentos restantes*

*Letras disponibles: bcdefghijklnopqrstuvwxyz*

*Por favor ingresa una letra: e*

*Oops! Esa letra no está en mi palabra: ma\_ \_*

*----------------------*

*Tienes 3 intentos restantes*

*Letras disponibles: bcdfghijklmnopqrtuvwxyz*

*Por favor ingresa una letra: e*

*Oops! Ya has intentado esa letra. Te quedan 0 advertencias: ma\_ \_*

*----------------------*

*Tienes 3 intentos restantes*

*Letras disponibles: bcdfghijklmnopqrtuvwxyz*

*Por favor ingresa una letra: e*

*Oops! Ya has intentado esa letra. No tienes más advertencias, así que pierdes un intento: ma\_ \_*

*----------------------*

*Tienes 2 intentos restantes*

*Letras disponibles: bcdefghijklmnopqrtuvwxyz*

*Por favor ingresa una letra: o*

*Buen intento: ma\_ o*

*----------------------*

*Tienes 2 intentos restantes*

*Letras disponibles: bcdefghijklmnpqrtuvwxyz*

*Por favor ingresa una letra: n*

*Buen intento: mano*

*----------------------*

*¡Felicitaciones, has ganado!*

*Tu puntaje total para este juego es: 8*

La salida de un juego **perdedor** debería verse así.

*Cargando lista de palabras desde archivo...*

*78566 palabras cargadas.*

*Bienvenido al juego de Ahorcados!*

*Estoy pensando en una palabra que tiene 4 letras*

*Tienes 3 advertencias restantes*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: abcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: e*

*Oops! Esa letra no está en mi palabra: \_ \_ \_ \_*

*----------------------*

*Tienes 4 intentos restantes*

*Letras disponibles: abcdfghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: b*

*Oops! Esa letra no está en mi palabra: \_ \_ \_ \_*

*----------------------*

*Tienes 3 intentos restantes*

*Letras disponibles: acdfghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: c*

*Oops! Esa letra no está en mi palabra: \_ \_ \_ \_*

*----------------------*

*Tienes 2 intentos restantes*

*Letras disponibles: adfghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: 2*

*Oops! Esa no es una letra válida. Te quedan 2 advertencias: \_ \_ \_ \_*

*----------------------*

*Tienes 2 intentos restantes*

*Letras disponibles: adfghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: d*

*Oops! Esa letra no está en mi palabra: \_ \_ \_ \_*

*----------------------*

*Tienes 1 intentos restantes*

*Letras disponibles: afghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: a*

*Buen intento: \_ \_ \_ a*

*----------------------*

*Tienes 1 intentos restantes*

*Letras disponibles: afghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: m*

*Oops! Esa letra no está en mi palabra: \_ \_ \_ \_a*

*----------------------*

*Lo siento, se te acabaron los intentos. La palabra era ‘hola’*

Una vez que haya completado y probado su código (donde ha proporcionado manualmente la palabra "secreta", dado que saberlo le ayuda a depurar su código), es posible que desee intentar ejecutarlo contra la computadora. Si se desplaza hacia abajo hasta la parte inferior del archivo que proporcionamos, verá dos líneas comentadas debajo del texto

if \_\_name\_\_ == "\_\_main\_\_":

# secret\_word = choose\_word(wordlist)

# hangman(secret\_word)

Estas líneas usan funciones que hemos proporcionado (cerca de la parte superior de ahorcado.py), que es posible que desee examinar. Intenta "descomentar" estas líneas y vuelve a cargar tu código. Esto le dará la oportunidad de probar sus habilidades con la computadora, que usa nuestras funciones para cargar un gran conjunto de palabras y luego elegir una al azar.

**PROBLEMA 4**

**EL AHORCADO PARTE 3: EL JUEGO CON PISTAS**

Si ha intentado jugar Ahorcado contra la computadora, habrá notado que no siempre es fácil ganarle a la computadora, especialmente cuando selecciona una palabra esotérica (como "esoterico"). Sería bueno que le pidieras a la computadora una pista, como una lista de todas las palabras que coinciden con lo que has adivinado.

Por ejemplo, si la palabra oculta es "manzana", y hasta ahora ha adivinado la letra "m" y la letra “a”, para que sepa que la solución es "m\_ \_ \_ \_ \_ a", donde necesita adivinar las 5 letras faltantes, podría ser bueno saber que el conjunto de palabras coincidentes (al menos en función de lo que cargó inicialmente la computadora) son:

*malsana, mangana, mariana, marrana, mediana, medrana, mengana, montana, morsana, mundana*

Vamos a crear una variación de Ahorcado (a esto lo llamamos *hangman\_with\_hints*, y le hemos proporcionado una estructura inicial para escribirlo), con la propiedad de que si adivina el carácter especial \* la computadora encontrará todas las palabras de su lista cargada que podría coincidir con su palabra actual adivinada, e imprimir cada uno de ellos. Por supuesto, no recomendamos probar esto en el primer paso, ya que esto imprimirá todas las 78,566 palabras que cargamos. Pero si te estás acercando a una respuesta y te estás quedando sin intentos, esto podría ser útil.

Para hacer esto, le pediremos que primero complete dos funciones auxiliares:

**3A) Coincidir con la palabra actual adivinada**

*match\_with\_gaps* toma dos parámetros: *my\_word* y *other\_word*. *my\_word* es una instancia de una palabra adivinada, en otras palabras, puede tener algunos \_ en lugares (como 'm\_ \_ o'). *other\_word* es una palabra normal en español.

Esta función debe devolver *True* si las letras adivinadas de *my\_word* coinciden con las letras correspondientes de *other\_word*. Debería devolver *False* si las dos palabras no son de la misma longitud o si una letra adivinada en *my\_word* no coincide con el carácter correspondiente en *other\_word*.

Recuerde que cuando se adivina una letra, su código revela todas las posiciones en las que aparece esa letra en la palabra secreta. Por lo tanto, la letra oculta (\_) no puede ser una de las letras de la palabra que ya se ha revelado.

**Ejemplo de Uso:**

*>>>* match\_with\_gaps *=* ("ma\_ o", "gano")

*False*

*>>>* match\_with\_gaps *=* ("ju\_ \_ o", "banana")

*False*

*>>>* match\_with\_gaps *=* ("\_ e\_ \_ il", "textil")

*True*

*>>>* match\_with\_gaps *=* ("te\_ \_ il", "textil")

*False*

**Consejo**: es posible que desee utilizar strip () para deshacerse de los espacios en la palabra para comparar longitudes

**3B) Mostrando todas las coincidencias posibles**

*show\_possible\_matches* toma un solo parámetro: *my\_word* que es una instancia de una palabra adivinada, en otras palabras, puede tener algunos \_s en lugares (como 't\_ \_ t').

Esta función debe imprimir todas las palabras en *wordlist* (observe dónde hemos definido esto al principio del archivo, línea 51) que coincidan con my\_word. Debería imprimir "No se encontraron coincidencias" si no hay coincidencias.

**Ejemplo de Uso:**

*>>> show\_possible\_matches =* ("m\_ \_ \_ \_ \_ a")

*malsana, mangana, mariana, marrana, mediana, medrana, mengana, montana, morsana, mundana*

*>>> show\_possible\_matches =* ("mannn \_")

*No se encontraron coincidencias*

*>>> show\_possible\_matches =* ("\_ uerta")

*huerta, muerta, tuerta*

**3C) Ahorcado con pistas**

Ahora debería poder replicar el código que escribió para ahorcado como el cuerpo de *hangman\_with\_hints*, luego hacer una pequeña adición para permitir que el usuario adivine un asterisco (\*), en cuyo caso la computadora imprimirá todas las palabras que coinciden con ese intento.

El usuario no debe perder un intento si el carácter ingresado es un asterisco.

Comenta las líneas de código que usaste para jugar al juego original de Ahorcado:

*secret\_word = choose\_word (wordlist)*

*hangman(secret\_word)*

Y elimine el comentario de estas líneas de código que hemos proporcionado en la parte inferior del archivo para reproducir su nuevo juego de Ahorcado con pistas:

*#secret\_word = choose\_word (wordlist)*

*#hangman\_with\_hints (secret\_word)*

**Ejemplo de código:**

La salida después de ingresar un asterisco debe verse como se muestra a continuación. Todos los demás resultados deben seguir el juego Ahorcado descrito en la Parte 2 anterior.

*Cargando lista de palabras desde archivo...*

*78566 palabras cargadas.*

*Bienvenido al juego de Ahorcados!*

*Estoy pensando en una palabra que tiene 7 letras*

*Tienes 3 advertencias restantes*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: abcdefghijklmnopqrstuvwxyz*

*Por favor ingresa una letra: m*

*Buen intento: m\_ \_ \_ \_ \_ \_*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: abcdefghijklnopqrstuvwxyz*

*Por favor ingresa una letra: a*

*Buen intento: ma \_ \_ a\_ a*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: bcdefghijklnopqrstuvwxyz*

*Por favor ingresa una letra: \**

*Las posibles coincidencias de palabras son:*

*malsana, mangana, manzana, mariana, marrana*

*----------------------*

*Tienes 6 intentos restantes*

*Letras disponibles: bcdefghijklnopqrstuvwxyz*

*Por favor ingresa una letra: z*

*Buen intento: ma \_ za\_ a*

**¡Esto completa el problema de aplicación!**