



Proyecto I - Parte I

Juego: ENCUENTRA SU PAR

CS1111 - Programación 1
Laboratorio 1.03

Integrantes

Ormeño Gonzales, Ernesto Paolo
Aquino Espinoza, Juan Leibniz
Rodriguez Camarena, Milagro Alessandra
Valverde Huaman, Max Hector
Pinedo Saldaña, Addy Valentina
Romero Huamaní, Antonio Jesus

Índice

Índice	2
Resumen	3
Introducción	4
Definiciones:	5
Funcionamiento del Programa	6
Ejemplo de ejecución	18
Conclusiones	21
Recomendaciones	22

Resumen

- **Tipos de datos:** Los tipos de datos más comunes en Python son strings(str), enteros (int, ej.:1,2,3,4...), flotantes (float, ej.: 1.25, 1.45, 6/9) y booleanos (bool, ej.:True o False).
- **Variables:** Una variable almacena diferentes tipos de datos que se pueden cambiar/actualizar a medida que se ejecuta el programa.
- **Strings:** En Python, un string es una cadena de caracteres representada entre comillas. Por ejemplo: "Hola", "1", "a", etc.
- **Funciones:** Una función es un conjunto de instrucciones para realizar una tarea específica. Asimismo, recibe parámetros y devuelve un resultado. Para este proyecto, las funciones se invocarán desde los módulos.
- **Listas:** Las listas son estructuras de datos mutables con múltiples valores. Por ejemplo: lista_1 = [1,2,3,4,5], lista_2 = ["hola", "como", "estas"].
- **Matriz:** Las matrices son listas de listas, de tipo bidimensional, por ejemplo:
 - matriz = [[1,2,3,4,5], [6,7,8,9], [10,11,12,13]]
- **Diccionarios:** Un diccionario es una colección de datos que almacena datos en el formato: clave:valor, tal es el caso de: d = {1: "número", 2: "letra", 3: ["a", "b", "c"]}
- **Módulos:** Los módulos son extensiones .py que implementan funciones. Estos pueden ser importados desde otro módulo mediante el "import".
- **Ciclos:** El ciclo while y for son una estructura que permite repetir un conjunto de código. Sin embargo, el ciclo while se repite hasta una determinada condición, mientras que el otro recorre una estructura iterable.

Introducción

Python es un lenguaje de programación útil y sencillo en comparación con otros lenguajes. Su orientación permite el desarrollo web y de aplicaciones informáticas. Por tal motivo en este proyecto “*Encuentra su par*” se puso en práctica los conocimientos adquiridos sobre variables booleanas, bucles anidados, librerías (random) y, especialmente, matrices de Python. Es así que en palabras simples, las matrices se pueden definir como listas de listas que permiten almacenar valores de tipo bidimensional. No obstante, surge la siguiente pregunta de investigación: **¿De qué manera utilizar matrices para programar un juego que halle un par de letras aleatorias en Python?**

Por esta razón, el objetivo de este proyecto es crear un programa que halle un par de letras aleatorias empleando matrices en Python. Para ello, se inspiró en el conocido juego “Memoria”, que, además, puede ser practicado desde niños hasta adultos y de gran utilidad en pruebas psicológicas para buscar quien tiene la mayor memoria para encontrar todos los pares de letras.

Definiciones:

Módulos:

- errors.py (Errores):
Este módulo contiene las funciones que controlan los errores del programa. Primero, valida que las filas no sean menores a 2 y que las casillas (filas*columnas) sean menores que 24, caso contrario imprime: `El número de filas debe ser mayor a 1 El total de casilleros ({boxes}) debe ser par y menor que 25`. Segundo, válida si se ingresa una posición fuera del tablero del juego, caso contrario imprime: `'El casillero no existe, verifique la casilla que desea abrir!'`. Tercero, válida si se ingresa el casillero que ya está abierto, caso contrario imprime: `'Ese casillero está abierto actualmente'`. Cuarto, válida si el casillero ya fue abierto, es decir, si una letra ya fue abierta. `Ese casillero ya fue abierto`
- success.py (Éxitos):
Este módulo contiene las funciones que implementan los mensajes de victoria. Primero, cuando se encuentra el par de la letra, imprime: `{letter} HA SIDO ENCONTRADA!`. Segundo, cuando el juego se acabó, imprime `"FIN DEL JUEGO! GANASTE!"`.
- table.py (Tabla):
Este módulo contiene las funciones que crean las listas base y de asteriscos, la función que rellena esas listas base con pares de letras aleatorias, la función que imprime el tablero de juego y la función que muestra los pares de letras en el tablero de juego.
- menu.py (Menú):
Este módulo contiene la presentación del juego y pide las filas y columnas para iniciar el juego y en lo que se basará el tablero de juego.
- Juego.py:
Este es el módulo principal que importa el resto de módulos para ejecutar el juego.

Funcionamiento del Programa

1. Menú Principal (archivo menu.py)

Para el menú principal se crearon tres funciones, una para solicitar las filas y columnas de la tabla, otra para pedir que el usuario abra un casillero, y para imprimir la intro de la interfaz.

Para la primera, se pide al usuario ingresar el número de filas y columnas, y se evalúa que los requisitos para aceptar dichos parámetros sea verdadero, caso contrario se vuelve a pedir el ingreso de los datos.

```
# Definimos una función para solicitar al usuario el número de filas y columnas de la tabla.
def ask_dimensions():
    # Definimos las filas para posteriormente solicitarlas al usuario
    rows = None

    # Definimos las columnas para posteriormente solicitarlas al usuario
    columns = None

    # Definimos el estado de la validación
    validated = False

    # Mientras que las filas y columnas se tengan que validar, se le solicita nuevamente al usuario que las ingrese
    while not validated:
        # Pedimos la usuario que ingrese el número de filas de la tabla
        rows = int(input('Ingrese el número de filas: '))

        # Validamos que el número de filas sea mayor a 1, mediante la función definida en el módulos de errors
        rows_validation = errors.validate_rows_dimension(rows)

        # Si el número de filas no es mayor a 1, volvemos a empezar el ciclo while
        if not rows_validation:
            continue

        # Pedimos la usuario que ingrese el número de columnas de la tabla
        columns = int(input('Ingrese el número de columnas: '))

        # Validamos que la multiplicación entre filas y columnas sea par y menor o igual que 24, usando la función
        # del módulo errors
        validated = errors.validate_boxes_dimension(rows, columns)

    return rows, columns
```

El return son los parámetros de fila columna para utilizarlos posteriormente en la creación de las listas de juego.

```
Ingrese el número de filas: 0
El número de filas debe ser mayor a 1

Ingrese el número de filas: 3
Ingrese el número de columnas: 1
El total de casilleros (3) debe ser par y menor que 25

Ingrese el número de filas: 2
Ingrese el número de columnas: 2

Process finished with exit code 0
```

Ejecución de ask_dimensions:

1. Previamente, se importa el archivo errors y define las variables rows y columns igual a "None", y validated como "False".

2. En un while negando validated, se pregunta por las filas como “rows” y las columnas como “columns”.
3. Con la función validate_rows_dimension del archivo errors, se evalúa que se cumplan las condiciones del juego.
4. Finalmente, los valores de retorno serán las filas y columnas, rows y columns respectivamente.

La segunda función, ask_box_position, recibe como parámetros complete_table, que es la tabla con los valores de letras; opened_boxes, los casilleros abiertos; y un valor opcional actual_box_value que cuando es diferente de None significa que hay un casillero abierto.

```
# La función pide como parámetros la tabla con los valores reales para poder validar posteriormente si el casillero fue
# abierto y pide opcionalmente el casillero actual, valor útil cuando el usuario ya reveló una letra y está
# buscando su par
def ask_box_position(complete_table, opened_boxes, actual_box_value=None):
```

Seguidamente, definimos las variables input_row, la fila del casillero a abrir; input_column, la columna del casillero a abrir, y validation, que va a mencionar si se cumplieron las validaciones.

```
# Definimos la fila ingresada
input_row = None

# Definimos la columna ingresada
input_column = None

# Definimos la validación de la entrada
validation = False
```

Luego, creamos un ciclo while que se repite hasta que ya no se necesite validar los datos porque están correctos. Dentro de este ciclo, le pedimos al usuario que ingrese la fila y columna del casillero por abrir, con el formato fila,columna; el valor es almacenado en position. Con position, definimos input_row, dándole el carácter en la primera posición; también, definimos input_column, señalando el carácter en la tercera posición. Después, validamos que el casillero exista, no esté abierto o se encuentre abierto, mediante la función validate_input_position del módulo errors; cambiando el valor de validation por el retorno de la función.

```
# Mientras que el input no cumpla con la validación, se le pide al usuario que ingrese de nuevo los datos.
while not validation:
    # Pedimos al usuario que ingrese la fila y columna con el formato fila,columna del casillero que desea abrir
    position = input('Abra un casillero en formato fila,columna (ej. 1,2) >>> ')

    # Obtenemos el primer valor del input, que es la fila
    input_row = int(position[0])

    # Obtenemos el tercer valor del input, que es la columna
    input_column = int(position[2])

    # Validamos que la fila y columna existan, que no estén abiertas o que no se encuentre abierta
    validation = errors.validate_input_position(complete_table, input_row, input_column, opened_boxes,
                                                actual_box_value)
```

Una vez se sale del ciclo while, retornamos la fila y columna ingresada:

```
return input_row - 1, input_column - 1
```

Por último, está la función `print_interface_intro`:

- Se imprime un logo que dice “juego encuentra su par” como introducción al juego.
- Y se imprimen las indicaciones del juego, es decir, como se deben ingresar el número de filas y columnas.
- Por último, el mensaje de advertencia de la condición para el número de casillas.

```
def print_interface_intro():
    # Imprimimos el nombre del juego con letras de color amarillo
    print(f'\x1b[1;33m' * 88)
    print("""
JUEGO ENCUENTRA SU PAR
""")
    print(' ' * 88 + '\x1b[1;36m')

    # Imprimimos las indicaciones del juego con letras de color cyan
    print('Ingrese el número de filas y columnas del juego')
    print('El total de casilleros (filas x columnas) debe ser par! \n' + '\x1b[0;0m')
```

La ejecución de la función se vería de la siguiente manera:

```
*****
JUEGO ENCUENTRA SU PAR
*****
Ingrese el número de filas y columnas del juego
El total de casilleros (filas x columnas) debe ser par!

Process finished with exit code 0
```

2. Matrices del juego (archivo `table.py`):

Para crear las matrices del juego se crearon 3 funciones: una para crear las tablas, otra para rellenar las tablas y otra para mostrar el tablero de juego.

2.1 Función para crear la tabla(`create_table`):

Mediante una función que pida filas, columnas y el valor de un carácter, se genera una matriz cuyos valores son un carácter, tantas columnas y filas tenga. Como se muestra en el siguiente ejemplo:

```
# Definimos una función que crea una tabla según el número de filas y columnas que se le brinda. Además, cuenta con
# un parámetro char opcional, que es útil cuando queremos crear la tabla inicial con la que jugará el usuario
def create_table(rows, columns, char='0'):
    table = []
    for row in range(rows):
        table.append([char] * columns)
    return table
```

Ejemplo:

```
[[0, 0], [0, 0]]
[['*', '*'], ['*', '*']]

Process finished with exit code 0
```


2.2. Función para rellenar las tablas(fill_table):

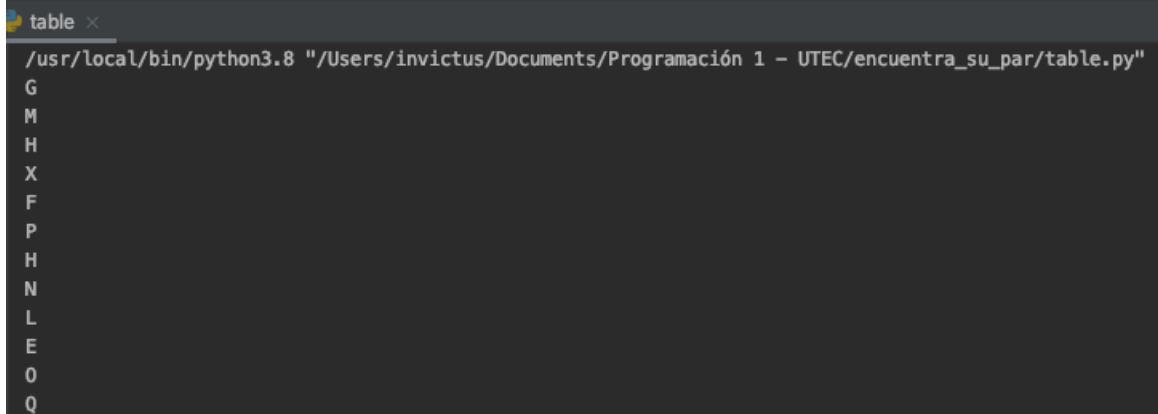
Luego de crear la lista de listas o matriz con ceros, se procede a rellenarla con valores de letras(A - Z) aleatorios. Primero, se define la cantidad de pares de letras que puede haber, por ello, se toma la mitad de las casillas (filas*columnas), ya que no buscamos el total de letras, sino pares de letras.

```
# Definimos una función para llenar la tabla con las letras
def fill_table(table):
    n_rows = len(table)
    n_columns = len(table[0])
    n_boxes = n_rows * n_columns

    # Recorremos la tabla en un rango de la mitad de sus casilleros, ya que debemos colocar una letra en dos casilleros
    for i in range(n_boxes // 2):
```

Luego, se escogen las letras aleatoriamente dentro de la lista de las letras, "abc", mediante la función choice, como se muestra en el siguiente ejemplo:

```
for i in range(24 // 2):
    # Escogemos una letra aleatoria
    letter = random.choice(abc)
    print(letter)
```



```
table x
/usr/local/bin/python3.8 "/Users/invictus/Documents/Programación 1 - UTEC/encuentra_su_par/table.py"
G
M
H
X
F
P
H
N
L
E
O
Q
```

Finalmente, en el juego, con un *ciclo while* de 0 a 1 creamos las iteraciones que van a rellenar la matriz. 1 par son 2 letras, por eso, son 2 iteraciones. Sin embargo, para ir rellenando la matriz (en un inicio con ceros), se va a ir evaluando aleatorias posiciones de la matriz, para que los pares de letras queden en diferentes posiciones. Por eso, se hace un *random.randint* entre 0 y el número de filas o columnas-1, pues se van a evaluar posiciones. Ahora, dentro de una variable *casilla_n* se va a ir guardando el **valor** de la matriz (inicialmente con ceros) en la posición aleatoria previamente definida. Luego se evalúa si el valor (no la posición) es cero y se actualiza con la letra aleatoria que se escogió. A continuación un ejemplo:

```

abc = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
casillas = filas*columnas

for i in range(casillas // 2): #cantidad de pares de letras
    letra = random.choice(abc) # escoger una letra aleatoria de la lista con el abecedario "abc"

    i = 0
    while i < 2: #el numero de iteraciones es 2 porque buscamos 1 par, osea 2 letras identicas
        random_filas = random.randint(0,filas-1) #0,1
        random_columna = random.randint(0,columnas-1) #0,1,2,3

        casilla_n = tabla_letras[random_filas][random_columna]

        if casilla_n == 0:
            tabla_letras[random_filas][random_columna] = letra #actualizacion de la tabla de letras para volverla una tabla de letras aleatorias
            i = i + 1
    abc.remove(letra)

```

```

W   escogida
F: 1 C: 3 -----
    en esta posicion es igual a 0
F: 1 C: 2 -----
    en esta posicion es igual a 0

D   escogida
F: 0 C: 1 -----
    en esta posicion es igual a 0
F: 0 C: 0 -----
    en esta posicion es igual a 0

L   escogida
F: 1 C: 3 -----
F: 1 C: 1 -----
    en esta posicion es igual a 0
F: 0 C: 0 -----
F: 1 C: 0 -----
    en esta posicion es igual a 0

K   escogida
F: 0 C: 2 -----
    en esta posicion es igual a 0
F: 1 C: 1 -----
F: 0 C: 3 -----
    en esta posicion es igual a 0

[['D', 'D', 'K', 'K'], ['L', 'L', 'W', 'W']]
D D K K
L L W W

Process finished with exit code 0

```

Para una matriz de ceros de 2 filas y 4 columnas, la cantidad de casilleros sería 8, entonces, habrían 4 pares de letras escogidas, en el ejemplo serían (W, D, L, K).

Luego se escogió posiciones aleatorias dentro del rango de sus filas y columnas. La primera posición siempre será la más fácil en evaluar, porque la matriz inicial empieza con ceros, y se le asigna el valor de la letra escogida. A medida que se le van asignando las letras, la evaluación o encontrar un cero se hace más larga, porque la matriz ya se está relleno. Por eso, rellenar la última letra toma más iteraciones o evaluaciones.

Al final, la matriz de ceros se volvió una matriz con pares de letras aleatorias.

Siendo el código en el juego:

```

# Definimos una función para llenar la tabla con las letras
def fill_table(table):
    n_rows = len(table)
    n_columns = len(table[0])
    n_boxes = n_rows * n_columns

    # Recorremos la tabla en un rango de la mitad de sus casilleros, ya que debemos colocar una letra en dos casilleros
    for i in range(n_boxes // 2):
        # Escogemos una letra aleatoria
        letter = random.choice(abc)

        i = 0

        # Creamos un ciclo while que buscará dos casilleros aleatorios donde colocar la letra seleccionada
        while i < 2:
            # Escogemos una fila y columna aleatoria
            random_row = random.randint(0, n_rows - 1)
            random_column = random.randint(0, n_columns - 1)

            # Obtenemos el valor en esa fila y columna
            box = table[random_row][random_column]

            # Si el valor es igual a 0, valor con el que se crea la tabla base, entonces podemos cambiarlo por la letra,
            # porque significa que no ha sido modificado previamente
            if box == 0:
                table[random_row][random_column] = letter
                i += 1

        # Una vez agregamos la letra, la quitamos de la lista para no repetirla
        abc.remove(letter)

    return table

```

2.3. Función que imprime el tablero del juego (print_table):

En esta parte se ingresa una tabla y luego se extrae las filas y columnas de la tabla para iterarlas.

Ejemplo:

```

def imprimir_tabla_juego(tabla):
    f = len(tabla) #filas
    c = len(tabla[0]) #columnas

    print(" ", end=" ") #espacio sobre la primera fila
    for i in range(1, c + 1):
        print(i, end=" ") #imprime las posiciones del tablero en la fila 0
    print()

    for i in range(f): #imprime la fila 1
        print(i + 1, end=" ")
        for j in range(c): #a continuacion se imprime los valores de la tabla en la posicion i(fila) y columna(j)
            print(tabla[i][j], end=" ")
        print()

    imprimir_tabla_juego() > for i in range(f)

```

juego x IDEA x

El total de casilleros (filas x columnas) debe ser par!

Ingrese el numero de filas : 2
 Ingrese el numero de columnas : 4

	1	2	3	4
1	*	*	*	*
2	*	*	*	*

En la primera parte, el primer `i` itera las filas guía y el segundo `i` itera las filas desde donde empieza la matriz. Por eso, primero imprime el número y luego tantos asteriscos como las columnas.

3. Errores (archivo `errors.py`):

A fin de mostrar los errores, creamos la función `validate_rows_dimension` que se encargará de verificar si el número de filas es mayor a 1, si lo es nos otorgará un `return` verdadero; si no, imprimirá un error que dirá *El número de casillas debe ser mayor a 1*, para ello utilizamos la condicional `if` seguido de un `not`.

```
# Definimos una función para validar que el número de filas de la tabla sea mayor a 1
def validate_rows_dimension(rows):
    if not 1 < rows:
        # Si el número de filas de la tabla no es mayor a 1, mostramos un mensaje de error
        print(format_error('El número de filas debe ser mayor a 1 \n'))
        return False
    return True
```

El número de filas debe ser mayor a 1

Process finished with exit code 0

Asimismo, empleamos una función similar, pero esta vez con respecto a la cantidad de filas y columnas, cuyo producto de estas (que da la cantidad de elementos que tiene) llamaremos *boxes* (que en sí representa los casilleros de la tabla).

```
# Definimos una función para validar que la multiplicación entre el número de filas y columnas de la tabla,
# los casilleros, sea par y menor igual que 24
def validate_boxes_dimension(rows, columns):
    # Obtenemos el número de casilleros
    boxes = rows * columns
```

Aquí le indicamos al programa que si *boxes* no es menor o igual a 24 o que no sea múltiplo de 2, se genere un `bool` `false` y que imprima en la consola *El total de casilleros (el número que haya ingresado el jugador) debe ser par y menor que 25*; de lo contrario será considerado como un `bool` `true`.

```
boxes = rows * columns

# Validamos que sean menor o igual que 24 y par
if not boxes <= 24 or boxes % 2 != 0:
    print(format_error(f'El total de casilleros ({boxes}) debe ser par y menor que 25 \n'))
    return False
return True
```

El total de casilleros (3) debe ser par y menor que 25

Process finished with exit code 0

Por otro lado, creamos una nueva función llamada `validate_input_position` que se encarga de verificar si el casillero que el jugador quiere abrir existe dentro de *boxes*, que no lo haya abierto ya o esté abierto en ese momento. Después de haber obtenido la cantidad de filas y columnas, validaremos si el casillero por abrir

se encuentra en el rango determinado antes. Para ello usaremos las variables *n_rows* (número de filas) y *n_columns* (número de columnas) para poder identificar el largo de estas, respecto a los valores *len(complete_table)* y *len(complete_table[0])*, parámetros pasados a la función en el archivo juego.py. Luego, con las variables *possible_box* y *possible_box_value* comprobaremos si el casillero ya ha sido abierto o está abierto. De ser así le aparecerá un mensaje de error en blanco definido como *error_message* que más adelante tomará otro valor dependiendo del error que se haya cometido.

```
# Definimos una función para validar que el casillero que el usuario quiere abrir existe, no haya sido abierto o
# esté abierto
def validate_input_position(complete_table, input_row, input_column, opened_boxes,
                           actual_box_value):
    # Obtenemos las filas y columnas de la tabla en juego, para posteriormente validar si el casillero por abrir
    # se encuentra en ese rango
    n_rows = len(complete_table)
    n_columns = len(complete_table[0])

    # Definimos el casillero por abrir, para luego validar si ya ha sido abierto
    possible_box = (input_row - 1, input_column - 1)

    # Obtenemos la letra del casillero por abrir, para luego validar si se el casillero se encuentra actualmente abierto
    possible_box_value = complete_table[input_row - 1][input_column - 1]

    # Definimos el mensaje de error en blanco
    error_message = ''
```

En esta parte lo que haremos que el mensaje de error varíe dependiendo de cuál se cometió, utilizaremos el condicional *if* para añadir la primera condición que trata sobre si el casillero no llegara a existir en la tabla, haciendo que *error_message* tome el valor de “El casillero no existe, verifique la casilla que desea abrir”. De igual manera, creamos un *elif* por si no se cometió ese error, pero sí el que si esa casilla ya ha sido abierta, para ese caso el *error_message* tomará el valor de *Ese casillero está abierto actualmente*. Nuestro último *elif* será para validar si el casillero ya ha sido abierto, en este caso *error_message* tomará el valor de *Ese casillero ya fue abierto*. Por otra parte, si no se comete ningún error obtendremos un bool *true*.

```
# Validamos si el casillero pertenece a la tabla
if not input_row <= n_rows or not input_column <= n_columns:
    # Cambiamos el mensaje de error
    error_message = 'El casillero no existe, verifique la casilla que desea abrir!\n'
# Validamos que el casillero no se encuentre abierto, viendo si hay algún casillero abierto y comparando su
# valor con el que se intenta abrir y si este ya ha sido abierto
elif actual_box_value is not None and actual_box_value == possible_box_value and possible_box in opened_boxes:
    # Cambiamos el mensaje de error
    error_message = 'Ese casillero está abierto actualmente!\n'
# Validamos si el casillero ha sido abierto
elif possible_box in opened_boxes:
    # Cambiamos el mensaje de error
    error_message = 'Ese casillero ya fue abierto!\n'
else:
    return True
```

El casillero no existe, verifique la casilla que desea abrir!

Process finished with exit code 0

```
Ese casillero está abierto actualmente
```

```
Process finished with exit code 0
```

```
Ese casillero ya fue abierto
```

```
Process finished with exit code 0
```

Para esta sección, asignaremos que se muestre en la consola el error correspondiente con un `print` en caso el `return` haya sido `false`. Por último, creamos la función `format_error` para que se muestren los mensajes de error en color rojo.

```
# Imprimimos el mensaje de error con el formato correspondiente
print(format_error(error_message))
return False

# Definimos una función para dar formato a los mensajes de error, color rojo
def format_error(error):
    return '\x1b[1;31m' + error + '\x1b[0;0m'
```

4. Éxitos (archivo success.py):

En este caso creamos 3 funciones para imprimir mensajes de éxito:

- En la primera, definimos la función “letter_found” para señalar que cierta letra ha sido encontrada, dándole a conocer a la función la letra mediante el parámetro `letter`.

```
# Definimos una función para imprimir que cierta letra ha sido encontrada
def letter_found(letter):
    print(format_success(f'{letter} HA SIDO ENCONTRADA!\n'))
```

```
A HA SIDO ENCONTRADA!
```

```
Process finished with exit code 0
```

- Para la segunda, creamos la función “win”, que sirve para indicarle al jugador que ya ganó el juego:

```
# Definimos una función para decirle al usuario que ha ganado el juego
def win():
    print(format_success('FIN DEL JUEGO! GANASTE!\n'))
```

```
FIN DEL JUEGO! GANASTE!
```

```
Process finished with exit code 0
```

- Por último, creamos la función “format success” para darle formato a los mensajes de éxito y salgan de color verde claro.

```
# Definimos una función para dar formato a los mensajes de éxito, color verde claro
def format_success(success):
    return '\x1b[1;32m' + success + '\x1b[0;0m'
```

5. Juego en sí (archivo juego.py):

Por último, en el archivo juego.py se encuentra toda la lógica del juego. Por ello, importamos los módulos menu, table y success, para poder establecerla.

```
# Importamos los módulos correspondientes
import menu
import table
import success
```

Además, se define una variable global, opened_boxes, lista que almacenará los casilleros abiertos.

```
# Definimos la lista de casilleros abiertos
opened_boxes = []
```

En esta sección solo se definió la función run, en esta está se encuentra el qué funciones ejecutar de acuerdo a la interacción del usuario. Primero, imprimimos la intro de la interfaz del juego, con la función print_interface_intro del módulo menu, y solicitamos que se ingresen las filas y columnas de la tabla a crear, con la función ask_dimensions del mismo módulo:

```
# Función donde se define la lógica del juego
def run():
    # Se imprime la intro de la interface del juego Encuentra su Par
    menu.print_interface_intro()

    # Pedidos al usuario las dimensiones de la tabla a jugar
    n_rows, n_columns = menu.ask_dimensions()
```

Luego, usando el módulo table, generamos una base_table, que posteriormente será llenada con las letras, de acuerdo al número de filas y columnas ingresados. Así como, una tabla de las mismas dimensiones, pero cuyos valores son "*", esta será la tabla oculta, hidden_table, con la que se jugará. Ahora, debemos llenar la base_table con las letras, mediante la función fill_table. Por último, imprimimos la hidden_table al usuario.

```
# Creamos la tabla base para la tabla con letras aleatorias
base_table = table.create_table(n_rows, n_columns)

# Creamos la tabla con *, para que el usuario no conozca los valores reales
hidden_table = table.create_table(n_rows, n_columns, '*')

# Llenamos la tabla base con letras aleatorias
complete_table = table.fill_table(base_table)

# Imprimimos la tabla con los valores ocultos (*)
table.print_table(hidden_table)
```

Seguidamente, creamos un ciclo while que se va a iterar hasta que la hidden_table, que es con la que juega el usuario, no sea igual a la complete_table, la tabla con

las letras; ya que, mientras que no sean iguales, significa que no las ha encontrado todas y por ende aún no gana. Dentro del ciclo, se pedirá que se abra un casillero almacenando la fila y columna en `input_row` e `input_column`. Una vez obtenido los datos se revelará la letra en esa posición, imprimiendo la tabla con la función `show_element_table` del módulo `table`. De ahí, se agregará dicho casillero a los abiertos. Ahora el usuario debe encontrar el par de la letra, por lo que almacenamos la casilla actual en `actual_box` y declaramos que la pareja todavía no ha sido encontrada en `pair_found`.

```
# Mientras que la tabla con la que juega el usuario no sea igual a aquella con los valores reales,
# se sigue jugando porque significa que todavía no gana
while hidden_table != complete_table:
    # Pedimos al usuario que ingrese la fila y columna del casillero a abrir
    input_row, input_column = menu.ask_box_position(complete_table, opened_boxes)

    # Mostramos al usuario la tabla con los valores ocultos, revelando la letra del casillero que eligió
    table.show_element_table(hidden_table, complete_table, input_row, input_column)

    # Agregamos la fila y columna a la lista de casilleros abiertos, aquel que el usuario acaba de revelar
    opened_boxes.append((input_row, input_column))

    # Definimos el casillero en el que se encuentra el usuario, el casillero que acaba de abrir
    actual_box = (input_row, input_column)

    # Definimos si el par de la letra se ha encontrado
    pair_found = False
```

Mientras que la pareja no sea encontrada, se guardará el valor del casillero abierto y se le solicitará nuevamente que ingrese una posición en la tabla, pero esta vez a la función `ask_box_position` le pasamos `actual_box_value`, que será útil para validar la entrada del usuario.

```
# Mientras que el par de la letra no se encuentre, le seguiremos pidiendo al usuario que ingrese otra
# posición de fila y columna para abrir
while not pair_found:
    # Obtenemos la letra del casillero abierto
    actual_box_value = complete_table[actual_box[0]][actual_box[1]]

    # Pedimos al usuario que ingrese la fila y columna del casillero a abrir, y así comprobar si ha
    # encontrado al par de la letra
    input_row, input_column = menu.ask_box_position(complete_table, opened_boxes, actual_box_value)
```

A continuación, se guardará la posición del casillero nuevo en `possible_box` y su valor en `complete_table` dentro de `possible_box_value`. Estos datos nos servirán para conocer si se encontró la pareja de la letra, ya que comparamos su valor y si su posición la tabla es diferente. En caso sea así, se revelará la letra en pantalla, la guardamos en la variable `letter`, declaramos que fue encontrada, la añadimos a casilleros abiertos e imprimimos un mensaje de éxito pasando como parámetro `letter`. Caso contrario, simplemente se imprimirá la tabla y se repetirá el ciclo.


```

# Definimos el casillero que acaba de abrir el usuario, donde podría encontrarse el par de la letra
possible_box = (input_row, input_column)

# Obtenemos la letra del casillero
possible_box_value = complete_table[possible_box[0]][possible_box[1]]

# Validamos si las letras son iguales y que la fila columna de los casilleros son diferentes,
# para así saber si el usuario encontro el par de la letra
if actual_box_value == possible_box_value and actual_box != possible_box:
    # Si el usuario encontro el par de la letra, la revelamos en la tabla
    table.show_element_table(hidden_table, complete_table, input_row, input_column)

    # Definimos una variable con la letra encontrada
    letter = actual_box_value

    # Declaramos que la letra fue encontrada, para terminar el ciclo while
    pair_found = True

    # Agregamos la fila y columna a la lista de casilleros abiertos
    opened_boxes.append((input_row, input_column))

    # Imprimimos un mensaje de que la letra fue encontrada exitosamente
    success.letter_found(letter)
else:
    # Si el usuario no encontro el par, volvemos a imprimir la tabla
    table.print_table(hidden_table)

```

Una vez el usuario encuentra todas las letras, se termina el ciclo y se imprime el mensaje que ganó:

```

# Cuando termina el ciclo while, significa que el usuario ganó, por lo que imprimimos un mensaje de éxito
success.win()

```

Ejemplo de ejecución

Al ejecutar el archivo juego.py:

Se imprime la introducción de la interfaz y se solicita al usuario ingresa el número de filas y columnas de la tabla a jugar:

```
*****
JUEGO ENCUENTRA SU PAR
*****
Ingrese el número de filas y columnas del juego
El total de casilleros (filas x columnas) debe ser par!
Ingrese el número de filas: 
```

Primero, se pide ingresar el número de filas, el cual debe ser mayor a 1. En caso sea menor, se imprime un mensaje de error y se solicita de nuevo:

```
Ingrese el número de filas: 0
El número de filas debe ser mayor a 1
Ingrese el número de filas: 
```

Si el número ingresado es mayor a 1, se pide al usuario que ingrese el número de columnas:

```
Ingrese el número de filas: 3
Ingrese el número de columnas: |
```

Cuando el usuario ingresa el número de columnas, se hace una validación de que la multiplicación entre las filas y las columnas, los casilleros de la tabla, sea par y menor o igual que 24. Si la condición no se cumple, se imprime un mensaje de error:

```
Ingrese el número de filas: 3
Ingrese el número de columnas: 1
El total de casilleros (3) debe ser par y menor que 25
```

Posteriormente, se solicita ingresar de nuevo el número de filas y columnas, si se pasa la validación, se imprime la tabla correspondiente. Asimismo, el usuario ahora puede abrir un casillero:

```
Ingrese el número de filas: 3
Ingrese el número de columnas: 2

      1      2
1      *      *
2      *      *
3      *      *

Abra un casillero en formato fila,columna (ej. 1,2) >>> 
```

El casillero abierto por el usuario se revela en la tabla y se solicita abrir otro hasta que se encuentre la letra. Al encontrar la letra, aparece un mensaje de éxito y se pide abrir otro casillero para buscar la siguiente letra:

Abra un casillero en formato fila,columna (ej. 1,2) >>> 1,1

	1	2
1	M	*
2	*	*
3	*	*

Abra un casillero en formato fila,columna (ej. 1,2) >>> 3,1

	1	2
1	M	*
2	*	*
3	*	*

Abra un casillero en formato fila,columna (ej. 1,2) >>> 2,2

	1	2
1	M	*
2	*	M
3	*	*

M HA SIDO ENCONTRADA!

Abra un casillero en formato fila,columna (ej. 1,2) >>>

Cabe mencionar que cuando se intenta abrir un casillero, se valida si este existe, ya fue abierto o se encuentra abierto:

Abra un casillero en formato fila,columna (ej. 1,2) >>> 1,2

	1	2
1	M	S
2	*	M
3	*	*

Abra un casillero en formato fila,columna (ej. 1,2) >>> 4,4
El casillero no existe, verifique la casilla que desea abrir!

Abra un casillero en formato fila,columna (ej. 1,2) >>> 1,2
Ese casillero está abierto actualmente

Abra un casillero en formato fila,columna (ej. 1,2) >>> 2,2
Ese casillero ya fue abierto

El juego continúa hasta que el usuario encuentre todas las letras. Una vez las encuentra, se muestra un mensaje que ha ganado:

Abra un casillero en formato fila,columna (ej. 1,2) >>> 2,1

	1	2
1	M	S
2	S	M
3	*	*

S HA SIDO ENCONTRADA!

Abra un casillero en formato fila,columna (ej. 1,2) >>> 3,1

	1	2
1	M	S
2	S	M
3	U	*

Abra un casillero en formato fila,columna (ej. 1,2) >>> 3,2

	1	2
1	M	S
2	S	M
3	U	U

U HA SIDO ENCONTRADA!

FIN DEL JUEGO! GANASTE!

Conclusiones

1. Primero, comprobamos la relevancia que tienen las matrices en la representación de datos. En este caso, utilizamos matrices bidimensionales para la creación de tablas, teniendo que crearlas desde cero, llenarlas con letras e imprimirlas. Creemos que este tipo de datos brindan una gran versatilidad al momento de desarrollar programas donde los datos se encuentran en cierta posición, como es el caso de las filas y columnas; y creemos que hubiera sido más complejo elaborar el juego implementando otro tipo de estructuras, como solo variables de strings.
2. También, concluimos que al usar módulos, encapsulando el código, es más sencillo definir la lógica final del proyecto en un archivo principal. Tal y como lo hicimos en el juego, generamos archivos con cierta finalidad como: menú, errores, éxitos y tabla, donde definimos funciones bajo ese objetivo para después implementarlas de manera rápida en juego.py según la lógica del juego.
3. Tercero, mientras íbamos desarrollando el juego, caímos en cuenta de la importancia de comentar el código. Si bien es cierto, existe una planificación previa a escribirlo, cuando programamos muchas veces avanzamos y entendemos lo que hacemos porque estamos escribiendo de manera secuencial. No obstante, cuando otra persona lee el código, puede ser que no lo entienda porque hay funciones juntas o simplemente no comprende la lógica. Por eso, decidimos comentar la mayoría de las líneas que hicimos para que todos sepamos que hace cada cosa.
4. Asimismo, había ciertos detalles del juego, como el color de las letras que se imprimen, que desconocíamos. Por lo que, decidimos indagar un poco, encontrándonos con blogs sobre dicha cuestión. A partir de ello, caímos en cuenta de lo importante que es la investigación para un proyecto, así como es importante conocer el tema para poder comprender los conceptos que se usan en el contenido de las fuentes, para poder adaptarlo a nuestras necesidades.
5. Por último, la programación lleva tareas del mundo físico a la computadora mediante el uso de matemáticas. Si quisiéramos hacer el juego de manera física, sería rápido seguir las reglas o las validaciones, solo por lógica podríamos ver si se cumple o no algo; sin embargo, cuando queremos darle las instrucciones a la computadora para que ella se encargue de todo, debemos de pensar de manera más abstracta a base de números y comparaciones. Nos pareció interesante como existe un puente con las matemáticas en las actividades que desarrollamos.

Recomendaciones

1. En primer lugar, consideramos que hubiera sido más eficiente utilizar plataformas para programar cooperativamente, como GitHub o inclusive Replit. De esta manera, hubiéramos evitado tener varias versiones del código y compartir la última constantemente. Por lo que, para una siguiente oportunidad sería idóneo implementarlas.
2. Además, creemos que el tiempo jugó un papel importante, por temas de horarios entre los integrantes a veces era difícil reunirnos todos. La recomendación sería que desde el primer día que tengamos conocimiento del proyecto, compartir cuando estamos libres y planificar lo antes posible.
3. De igual manera, al programar nos dimos cuenta de que en diversas soluciones andábamos muchos ciclos, lo cual es ineficiente para el algoritmo. A pesar de que, escogimos la mejor que pudimos, consideramos que aún hay espacio de mejora.
4. Finalmente, había ocasiones donde alguno aceptaba algo que desconocía cómo hacerlo, pero a pesar de las complicaciones no pedía ayuda al grupo y se pasaba el tiempo. Consideramos que, para futuros proyectos, la sinceridad ante una dificultad es lo ideal, ya que de esa manera podemos avanzar más rápido y ayudarnos a mejorar entre todos.