



Proyecto I - Parte II

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

Resumen	3
Introducción	4
Definiciones	5
Módulo: Database.py	5
Funcionamiento del programa	6
Manipulación de fechas	6
Formato de la fecha	6
Función 1: get_current_date	6
Función 2: set_filename	7
Función 3: time_to_seconds	7
Función 4: subtract_dates	8
Formato JSON de letras e intentos	8
Función 1: letter_json_format	8
Función 2: attempt_json_format	9
Función 3: add_attempt	10
Función 4: add_letter	10
Manipulación del archivo JSON	10
Función 1: create_file	10
Función 2: read_file	11
Función 3: update_file	11
Resumen del juego y archivo histórico.csv	11
Función 1: calc_points	11
Función 2: get_game_summary	11
Función 3: update_history	12
Función 4: print_game_summary	13
Implementación de la base de datos en juego.py	14
Ejemplo de ejecución	16
Conclusiones	23
Recomendaciones	24

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.
- **Manipulación de archivos:** Para poder leer, escribir, añadir o ejecutar el contenido de un archivo, en Python tenemos la función open() que recibe la ruta del documento y la tarea a desarrollar.

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 listas. En la parte 1 de este proyecto se respondió a la siguiente interrogante: *¿De qué manera utilizar matrices para programar un juego que halle un par de letras aleatorias en Python?* No obstante, para acercarnos más a la realidad, nos preguntamos **¿De qué manera almacenar las letras e intentos del juego, la tabla, el tiempo y los puntos de cada usuario, usando JSON en Python, nos permite llevar un histórico de juegos pasados con el cual compararnos?**

Por esta razón, el objetivo de este proyecto es complementar nuestro programa con JSON para almacenar los datos del usuario con respecto al juego. De este modo, se desarrollaría un juego completo y al alcance de todos. 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

Módulo: Database.py

Principales funciones del módulo:

- `get_current_date`: función para establecer el formato Día-Mes-Año Hora-Mes-Segundo de la fecha actual.
- `set_filename`: función para implementar el nombre del archivo en el formato de la fecha.
- `create_file`: función para crear un archivo de Json.
- `read_file`: función para leer un archivo de Jason.
- `update_file`: función para actualizar un archivo de Json.
- `letter_json_format`: función para crear el diccionario anidado de cada letra del juego.
- `attempt_json_format`: función para crear el diccionario anidado por cada intento para hallar cada letra del juego.
- `add_attempt`: función para agregar cada intento al diccionario de intentos en la letra.
- `add_letter`: función para agregar cada letra al diccionario general.
- `update_history`: función para implementar el resto de funciones al juego.
- `time_to_seconds`: función para convertir el formato del tiempo a solo segundos.
- `subtract_dates`: función para calcular el tiempo total de duración del juego.
- `calc_points`: función para calcular los puntos del jugador.

Funcionamiento del programa

En el proyecto inicial de encuentra_su_par, para poder añadir las nuevas características, creamos una nueva carpeta llamada database donde guardamos el módulo database.py. El archivo posee distintas funciones:

- `get_current_date`, `set_filename`, `time_to_seconds` y `subtract_dates`: se trabaja con la librería `datetime`.
- `create_file`, `read_file` y `update_file`: se manipula el archivo JSON del juego.
- `letter_json_format`, `attempt_json_format`, `add_attempt` y `add_letter`: se da el formato JSON a las letras e intentos para luego guardarlo en el archivo.
- `update_history`, `get_game_summary` y `print_game_summary`: se genera el resumen del juego, se guarda en el histórico y se imprime en pantalla.
- `calc_points`: útil, junto a `time_to_seconds` y `subtract_dates`, para obtener el resumen del juego.

Manipulación de fechas

Antes de implementar las funciones de fechas, definimos el formato correspondiente:

Formato de la fecha

```
current_date_format = '%d-%m-%Y %H:%M:%S'
```

- Se importa la clase `datetime` para poder manejar y utilizar fechas y horas.
- Definimos la variable `'current_date_format'` en formato de día, mes, año, y horas, minutos y segundos

Función 1: `get_current_date`

```
def get_current_date():  
    current_date = dt.now()  
    formatted_current_date = current_date.strftime(current_date_format)  
  
    return formatted_current_date
```

- La función sirve para obtener la fecha y hora del intento.
- Empleamos la función `'now'` de `'datetime'`, de tal manera que obtenemos la fecha y hora actual. Se le asigna el valor a la variable `'current_date'`.
- Para cambiar el formato al definido con anterioridad, se usa la función `'strftime'`, asignándole este valor a `'formatted_current_date'` y retornándolo al final.

Función 2: set_filename

```
def set_filename():
    current_date = dt.now()
    filename = current_date.strftime('%d%m%Y%H%M%S')

    return filename

current_filename = set_filename()
file_path = f'./database/{current_filename}.json'
```

- La función retorna la variable 'filename', que servirá para crear la dirección del archivo de la base de datos del juego, 'file_path'.
- Con la función 'now' se extrae los valores de fecha y hora.
- Se da el formato solicitado con la función 'strftime', y al final asignando este valor a 'filename'.

Función 3: time_to_seconds

```
def time_to_seconds(str_datetime):
    datetime = dt.strptime(str_datetime, current_date_format)

    seconds = (60 * ((datetime.hour * 60) + datetime.minute)) + datetime.second

    return seconds
```

- Con esta función se obtienen los segundos transcurridos hasta cierta hora, 'str_datetime'.
- Como la variable de entrada es un string, debemos convertirla a un objeto datetime, eso lo logramos con 'strptime'.
- Finalmente, para obtener los segundos transcurridos, multiplicamos las horas por 3600, los minutos por 60 y lo sumamos a los segundos de la fecha inicial.

Función 4: subtract_dates

```
def subtract_dates(start_datetime, end_datetime):  
    start_datetime_seconds = time_to_seconds(start_datetime)  
    end_datetime_seconds = time_to_seconds(end_datetime)  
  
    dates_subtract = end_datetime_seconds - start_datetime_seconds  
  
    return dates_subtract
```

- La función retorna el tiempo de juego por intento.
- Las variables de ingreso son 'start_datetime' y 'end_datetime', hora inicial y hora final.
- Usando la función 'time_to_seconds' con la hora de inicio y hora final, se guarda en las variables 'start_datetime_seconds' y 'end_datetime_seconds'.
- Se realiza una simple resta y se obtiene el tiempo de juego, finalmente se lo asigna a la variable 'dates_subtract' que viene a hacer el valor retornado de la función.

Formato JSON de letras e intentos

Para el formato de letras e intentos se crearon 4 funciones. Las dos primeras para guardar las letras e intentos en diccionarios anidados y las 2 últimas para agregar los intentos a la letra y la letra al diccionario "database".


Función 1: letter_json_format

Para el formato de letras, se define a la función como "letter_json_format" que ingresa el valor de la letra y su posición. Primero, se define la variable "current_date" que almacena la fecha actual. Para esto, se utiliza la función "get_current_date".

Luego, se genera la variable "letter_dict" que guarda en un diccionario la letra como llave y en su valor otro diccionario la: "fechahora_apertura", "posicion_apertura" e "intentos".

Ahora, dentro de los valores de estas llaves se guarda: "current date", que es la fecha de inicio o apertura de esta letra. A modo de un diccionario, la posición de la fila y columna, para lo que, se pide los dos valores de la posición ingresada. Puesto que, la posición ingresada fue guardada como una tupla: (1,2), (3,4), etc.

Además, en intentos se guarda como un diccionario vacío. Ya que, luego se le irán agregando valores mientras se juega.



```
11 def letter_json_format(letter, position):
12     current_date = get_current_date() #una función dentro de otra función
13
14     letter_dict = {letter: { # formato donde por cada letra se registra f
15         "fecha_hora_apertura": current_date,
16         "posicion_apertura": {"f": position[0], "c": position[1]},
17         "intentos": {}
18     }}
19
20     return letter_dict
21
22 result_1 = letter_json_format("A", "12")
23 print(result_1)
24
25 result_2 = letter_json_format("C", "34")
26 print(result_2)
27
28 letter_json_format()
```

Run: prueba x

C:\Users\Usuario\AppData\Local\Programs\Python\Python310\python.exe "C:/Users/Usuario/Downloads/python/My practice/prueba.py"

```
{'A': {'fecha_hora_apertura': '29-06-2022 21:43:10', 'posicion_apertura': {'f': '1', 'c': '2'}, 'intentos': {}}}
```

```
{'C': {'fecha_hora_apertura': '29-06-2022 21:43:10', 'posicion_apertura': {'f': '3', 'c': '4'}, 'intentos': {}}}
```

Process finished with exit code 0

Así se muestra en el previo ejemplo de ejecución. En este caso, ingresamos la letra “A” y la posición “12”. También se intentó con la letra “C” y la posición “34”

Función 2: attempt_json_format

En esta parte la función "attempt_json_format" recibe los valores de entrada creando un diccionario con estos y con la fecha actual.

Si se encuentra el par de la letra, entonces se define una llave 'encontro' que recibe el valor del booleano, o sea True.

```
def attempt_json_format(attempts_count, position, pair_found):
    current_date = get_current_date()

    attempt_dict = {f"{attempts_count}": {
        "posicion": {"f": position[0], "c": position[1]},
        "fecha_hora_jugada": current_date
    }}

    if pair_found:
        attempt_dict['encontro'] = pair_found

    return attempt_dict
```

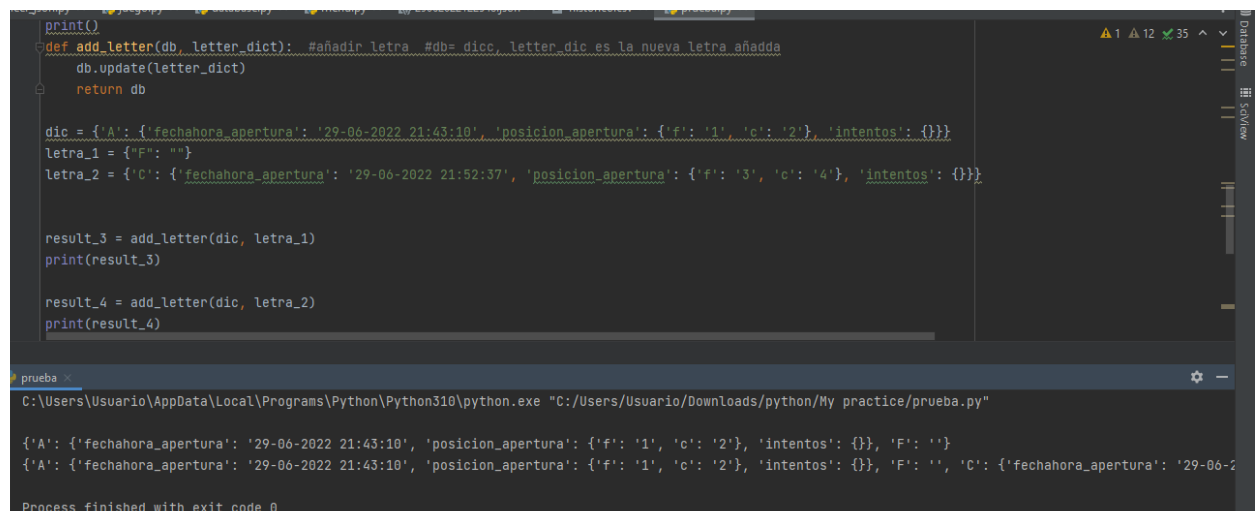
Función 3: add_attempt

Definimos otra función para adicionar un intento, se recibe la base de datos, la letra donde añadir el intento y se agrega este mediante la función update.

```
def add_attempt(db, letter, attempt):  
    db[letter]['intentos'].update(attempt)
```

Función 4: add_letter

Se crea la función “add_letter” que recibe como parámetros el diccionario en sí y la letra a añadir. Luego se agrega al diccionario la letra como si se tratara de una lista, solo que en vez de usar “append”, se emplea “update”.



```
print()  
def add_letter(db, letter_dict): #añadir letra #db= dict, letter dict es la nueva letra añadida  
    db.update(letter_dict)  
    return db  
  
dic = {'A': {'fecha_hora_apertura': '29-06-2022 21:43:10', 'posicion_apertura': {'f': '1', 'c': '2'}, 'intentos': {}}}  
letra_1 = {'F': ''}  
letra_2 = {'C': {'fecha_hora_apertura': '29-06-2022 21:52:37', 'posicion_apertura': {'f': '3', 'c': '4'}, 'intentos': {}}}  
  
result_3 = add_letter(dic, letra_1)  
print(result_3)  
  
result_4 = add_letter(dic, letra_2)  
print(result_4)  
  
prueba  
C:\Users\Usuario\AppData\Local\Programs\Python\Python310\python.exe "C:/Users/Usuario/Downloads/python/My practice/prueba.py"  
  
{'A': {'fecha_hora_apertura': '29-06-2022 21:43:10', 'posicion_apertura': {'f': '1', 'c': '2'}, 'intentos': {}}, 'F': ''}  
{'A': {'fecha_hora_apertura': '29-06-2022 21:43:10', 'posicion_apertura': {'f': '1', 'c': '2'}, 'intentos': {}}, 'F': '', 'C': {'fecha_hora_apertura': '29-06-2022 21:52:37', 'posicion_apertura': {'f': '3', 'c': '4'}, 'intentos': {}}}  
  
Process finished with exit code 0
```

Como se observa en este ejemplo de ejecución, a un diccionario que contiene la letra “A”, se le puede agregar otros diccionarios con llaves “F” o “C”, a través del update que implementa la función.

Manipulación del archivo JSON

Función 1: create_file

Generamos la función create_file la cual se encargará de originar el archivo y escribirá un diccionario / JSON vacío.

```
def create_file():  
    with open(file_path, 'w') as db:  
        db.write('{}')
```

Función 2: read_file

Luego, definimos la función `read_file` que se encarga de leer el archivo (utilizamos el encoding de utf-8, el más común) y retornará los datos JSON guardados en forma de diccionario.

```
def read_file():  
    with open(file_path, 'r', encoding='utf-8') as db:  
        data = json.load(db)  
  
    return data
```

Función 3: update_file

Por último, la función `update_file` que usaremos para actualizar el archivo, sobrescribiéndolo. El `json.dump` es para escribir los datos del diccionario Python a JSON y poder almacenarlos con dicho formato.

```
def update_file(new_db):  
    with open(file_path, 'w', encoding='utf-8') as db:  
        json.dump(new_db, db)
```

Resumen del juego y archivo histórico.csv

A fin de obtener el resumen del juego, se produjeron 3 funciones: `get_game_summary`, que retorna los datos importantes para actualizar el archivo histórico.csv; `calc_points`, que calcula los puntos del juego; `update_history`, que actualiza el histórico.csv, y `print_game_summary`, que imprime el resumen al terminar de jugar.

Función 1: calc_points

```
def calc_points(boxes, game_duration):  
    points = round(game_duration / boxes, 1)  
  
    return points
```

La función recibe el número de casillas y la duración del juego. A partir de estos, hace un cálculo simple y retorna el total de puntos ganados.

Función 2: get_game_summary

Esta toma como parámetros el número de columnas y filas con el que se jugó, valor que se retornan en el resumen y con los que se calculan el número de casillas.

```
def get_game_summary(rows, columns):
```

Luego, se lee el archivo JSON del juego y se halla la hora de inicio y fin del juego, mediante la primera letra abierta y el último intento hecho. De esta manera, con la función de `subtract_dates`, podremos tener la duración total.

```
db = read_file()

db_keys = list(db.keys())

start_datetime = db[db_keys[0]]['fechahora_apertura']

db_last_letter_attempts = db[db_keys[-1]]['intentos']
last_letter_attempts_keys = list(db_last_letter_attempts.keys())

end_datetime = db_last_letter_attempts[last_letter_attempts_keys[-2]]['fechahora_jugada']

game_duration = subtract_dates(start_datetime, end_datetime)
```

Seguidamente, se itera el diccionario de la base de datos y se cuenta la longitud de la llave `intentos` de cada letra. Así, podemos ir sumando hasta conseguir el número total de intentos.

```
attempts_count = 0
for values in db.values():
    attempts_count += len(values['intentos'])
```

Después, mediante la función `calc_point`, recibimos los puntos totales, pasándole como entrada la cantidad de casillas y la duración del juego.

```
boxes = rows * columns
points = calc_points(boxes, game_duration)
```

Finalmente, unimos todos nuestros datos en una plantilla separada por comas y los retornamos.

```
game_summary = f'{end_datetime},{rows},{columns},{attempts_count},{game_duration},{points}'

return game_summary
```

Función 3: `update_history`

Este código se encargará de actualizar el archivo `histórico.csv`. Además, recibe como parámetros las filas y columnas, datos necesarios para crear el resumen del juego.

```
def update_history(rows, columns):
    game_summary = get_game_summary(rows, columns)
    path = './database/historico.csv'
```

Una vez tenemos el resumen y la ruta del archivo. Abrimos el archivo en modo añadir y comprobamos si está vacío mediante la librería os, si este fuera el caso se le agregan los encabezados de la tabla. Posteriormente, se escribe el resumen obtenido.

```
with open(path, 'a') as history_file:
    if os.stat(path).st_size == 0:
        labels = "Fecha/hora del fin del juego,Número de filas,Número de columnas,Número de jugadas totales," \
                 "Tiempo total del juego (en segundos),Puntaje\n "
        history_file.write(labels)

    history_file.write(f'{game_summary}\n')
```

Por último, retornamos el resumen, porque será usado para imprimirlo al finalizar el juego.

```
return game_summary
```

Función 4: print_game_summary

La función recibe el resumen a imprimir, pero esta no mostrará todos los datos, sino solo el número de jugadas, tiempo y puntaje. Por lo que, se convierte el texto de resumen a lista y se emplea el slice de 3 a 6 para obtener los datos correspondientes. Además, producimos otra lista con los encabezados de cada uno.

```
def print_game_summary(summary):
    summary_array = summary.split(',')[3:6]
    columns = ['Número de jugadas', 'Tiempo (segundos)', 'Puntaje']
```

Imprimimos un texto de color amarillo que diga resumen:

```
print(f'\x1b[1;33m *' * 32)
print("""\x1b[1;33m
[Diagrama de un tablero de Go con piezas blancas y negras]
""")
print(f'\x1b[1;33m *' * 32)
```

Finalmente, iteramos los encabezados y el resumen, mostrándolos en pantalla con un margen izquierdo de 18.

```

for column in columns:
    print(f'\t\x1b[1;36m{column.ljust(18)}', end='')
print()

for element in summary_array:
    print(f'\t\x1b[1;32m{element.ljust(18)}', end='')
print()

```

Implementación de la base de datos en juego.py

Una vez tenemos definimos todas las funciones en el módulo de database, llegó la hora de implementarlo en el juego. Primero, lo importamos.

```
import database.database as db
```

Seguidamente, una vez pasamos todas las validaciones iniciales de la tabla a jugar, creamos el archivo JSON.

```

# 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()

    # 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)

    db.create_file()

```

Luego, cada vez que se busque una letra, leemos nuevamente la base de datos en busca de actualizaciones.

```
# 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:
    database = db.read_file()
```

Asimismo, cuando abrimos el primer casillero de la letra, generamos el formato JSON de la misma, la agregamos a la base de datos y la actualizamos.

```
# 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)

# Obtenemos la letra del casillero abierto
actual_box_value = complete_table[actual_box[0]][actual_box[1]]

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

letter_dict = db.letter_json_format(actual_box_value, actual_box)
db.add_letter(database, letter_dict)
db.update_file(database)
```

Mientras se esté buscando el par de la letra, definimos un contador, que nos servirá para saber el número de intentos en que se encuentra el jugador y almacenarlo a posterioridad en la base de datos.

```
attempts_count = 0
while not pair_found:
    attempts_count += 1
```

Tras finalizar el bucle de la casilla abierta en cuestión, generamos el formato JSON del intento, lo agregamos a la letra principal y actualizamos la base de datos.

```
attempt_dict = db.attempt_json_format(attempts_count, possible_box, pair_found)
db.add_attempt(database, actual_box_value, attempt_dict)
db.update_file(database)
```

Por último, cuando se termina el juego, actualizamos el archivo historico.csv, obtenemos el resumen del juego y lo imprimimos.

```
# Cuando termina el ciclo while, significa que el usuario ganó, por lo que imprimimos un mensaje de éxito
success.win()
game_summary = db.update_history(n_rows, n_columns)
db.print_game_summary(game_summary)
```

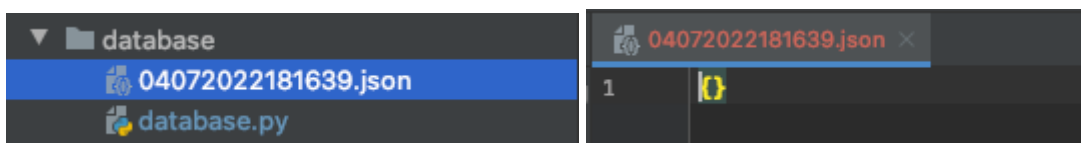
Ejemplo de ejecución

Al iniciar el juego y definir los parámetros para la tabla, se crea el archivo JSON vacío con el nombre correspondiente.

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

Ingrese el número de filas: 3
Ingrese el número de columnas: 3
  1 2
  1 * *
  2 * *
  3 * *

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

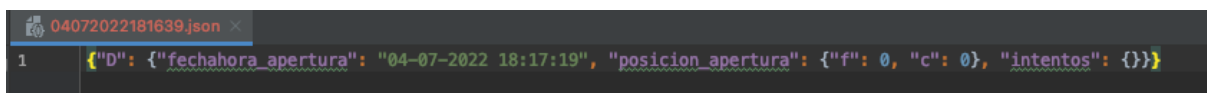


Tras abrir un casillero, se genera la llave de la letra y se almacena en el archivo.

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

  1 2
  1 D *
  2 * *
  3 * *

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



JSON:

```
{
  "D": {
    "fecha_hora_apertura": "04-07-2022 18:17:19",
    "posicion_apertura": {
      "f": 0,
      "c": 0
    },
    "intentos": {
    }
  }
}
```


Luego, mientras intentamos hallar el par, los intentos se van almacenando en la letra.

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

```
04072022181639.json
{
  "D": {
    "fechahora_apertura": "04-07-2022 18:17:19",
    "posicion_apertura": {
      "f": 0,
      "c": 0
    },
    "intentos": {
      "1": {
        "posicion": {
          "f": 2,
          "c": 0
        },
        "fechahora_jugada": "04-07-2022 18:18:46"
      }
    }
  }
}
```

JSON:

```
{
  "D": {
    "fechahora_apertura": "04-07-2022 18:17:19",
    "posicion_apertura": {
      "f": 0,
      "c": 0
    },
    "intentos": {
      "1": {
        "posicion": {
          "f": 2,
          "c": 0
        },
        "fechahora_jugada": "04-07-2022 18:18:46"
      }
    }
  }
}
```

Así hasta que se halla el par y se agrega el intento con la llave de “encontro” como True:

```
Abra un casillero en formato fila,columna (ej. 1,2) >>> 3,1
  1  2
1  D  *
2  *  *
3  G  *
Abra un casillero en formato fila,columna (ej. 1,2) >>> 2,1
  1  2
1  D  *
2  G  *
3  *  *
Abra un casillero en formato fila,columna (ej. 1,2) >>> 1,2
  1  2
1  D  D
2  *  *
3  *  *
D HA SIDO ENCONTRADA!
Abra un casillero en formato fila,columna (ej. 1,2) >>>
```

JSON:

```
{
  "D": {
    "fechahora_apertura": "04-07-2022 18:17:19",
    "posicion_apertura": {
```

```

        "f":0,
        "c":0
    },
    "intentos":{
        "1":{
            "posicion":{
                "f":2,
                "c":0
            },
            "fechahora_jugada":"04-07-2022 18:18:46"
        },
        "2":{
            "posicion":{
                "f":1,
                "c":0
            },
            "fechahora_jugada":"04-07-2022 18:20:25"
        },
        "3":{
            "posicion":{
                "f":0,
                "c":1
            },
            "fechahora_jugada":"04-07-2022 18:20:27"
        },
        "encuentro":true
    }
}
}

```

Se abre otra letra y se repite el proceso:

```

Abra un casillero en formato fila,columna (ej. 1,2) >>> 2,1
    1  2
  1  D  D
  2  G  *
  3  *  *
Abra un casillero en formato fila,columna (ej. 1,2) >>> 2,2
    1  2
  1  D  D
  2  G  E
  3  *  *
Abra un casillero en formato fila,columna (ej. 1,2) >>> 3,2
    1  2
  1  D  D
  2  G  *
  3  *  E
Abra un casillero en formato fila,columna (ej. 1,2) >>> 3,1
    1  2
  1  D  D
  2  G  *
  3  G  *
G HA SIDO ENCONTRADA!

```

JSON:

```

{
    "D":{

```

```

    "fechahora_apertura":"04-07-2022 18:17:19",
    "posicion_apertura":{
      "f":0,
      "c":0
    },
    "intentos":{
      "1":{
        "posicion":{
          "f":2,
          "c":0
        },
        "fechahora_jugada":"04-07-2022 18:18:46"
      },
      "2":{
        "posicion":{
          "f":1,
          "c":0
        },
        "fechahora_jugada":"04-07-2022 18:20:25"
      },
      "3":{
        "posicion":{
          "f":0,
          "c":1
        },
        "fechahora_jugada":"04-07-2022 18:20:27"
      },
      "encuentro":true
    }
  },
  "G":{
    "fechahora_apertura":"04-07-2022 18:23:24",
    "posicion_apertura":{
      "f":1,
      "c":0
    },
    "intentos":{
      "1":{
        "posicion":{
          "f":1,
          "c":1
        },
        "fechahora_jugada":"04-07-2022 18:23:26"
      },
      "2":{
        "posicion":{
          "f":2,
          "c":1
        },
        "fechahora_jugada":"04-07-2022 18:23:29"
      },
      "3":{
        "posicion":{
          "f":2,
          "c":0
        },
        "fechahora_jugada":"04-07-2022 18:23:32"
      },
      "encuentro":true
    }
  }
}

```

Una vez se halla la letra final, se imprime el resumen; se genera el archivo historico.csv, si este no existe, y se agregan los datos necesarios.

```
Abra un casillero en formato fila,columna (ej. 1,2) >>> 2,2  
    1   2  
1   D   D  
2   G   E  
3   G   *  
  
Abra un casillero en formato fila,columna (ej. 1,2) >>> 3,2  
    1   2  
1   D   D  
2   G   E  
3   G   E  
  
E HA SIDO ENCONTRADA!  
  
FIN DEL JUEGO! GANASTE!  
  
*****  
  
*****  
  
*****  
  
*****
```

Número de jugadas	Tiempo (segundos)	Puntaje
10	437	72.8

Process finished with exit code 0

JSON:

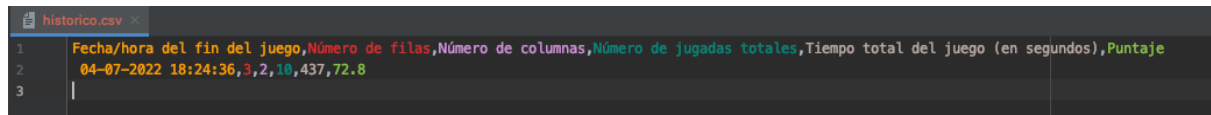
```
{
  "D": {
    "fechahora_apertura": "04-07-2022 18:17:19",
    "posicion_apertura": {
      "f": 0,
      "c": 0
    },
    "intentos": {
      "1": {
        "posicion": {
          "f": 2,
          "c": 0
        },
        "fechahora_jugada": "04-07-2022 18:18:46"
      },
      "2": {
        "posicion": {
          "f": 1,
          "c": 0
        },
        "fechahora_jugada": "04-07-2022 18:20:25"
      },
      "3": {
        "posicion": {
```

```

        "f":0,
        "c":1
    },
    "fecha_hora_jugada":"04-07-2022 18:20:27"
},
"encuentro":true
}
},
"G":{
    "fecha_hora_apertura":"04-07-2022 18:23:24",
    "posicion_apertura":{
        "f":1,
        "c":0
    },
    "intentos":{
        "1":{
            "posicion":{
                "f":1,
                "c":1
            },
            "fecha_hora_jugada":"04-07-2022 18:23:26"
        },
        "2":{
            "posicion":{
                "f":2,
                "c":1
            },
            "fecha_hora_jugada":"04-07-2022 18:23:29"
        },
        "3":{
            "posicion":{
                "f":2,
                "c":0
            },
            "fecha_hora_jugada":"04-07-2022 18:23:32"
        },
        "encuentro":true
    }
},
"E":{
    "fecha_hora_apertura":"04-07-2022 18:24:34",
    "posicion_apertura":{
        "f":1,
        "c":1
    },
    "intentos":{
        "1":{
            "posicion":{
                "f":2,
                "c":1
            },
            "fecha_hora_jugada":"04-07-2022 18:24:36"
        },
        "encuentro":true
    }
}
}

```

Archivo historico.csv:



Conclusiones

Tras el desarrollo del proyecto, pudimos llegar a las siguientes conclusiones:

1. Primero, el uso de módulos desde un inicio facilitó la nueva implementación. La incorporación de la base de datos al juego fue relativamente sencilla, la única tarea difícil fue analizar donde ejecutar cada función y que datos debíamos pasar como entrada. Caímos en cuenta que el correcto desarrollo de un proyecto, reduce la deuda técnica cuando se desea agregar algo.
2. También, aprendimos a trabajar con las librerías `datetime`, `json` y `os`, y comprendimos su ayuda al programar. Por ejemplo, `datetime` nos simplificó mucho el manipular cada aspecto de una fecha y hora, permitiéndonos conseguir los formatos que queríamos y poder trabajar con los campos como si fueran enteros. Asimismo, `Json` nos favoreció cuando debíamos leer y actualizar la base de datos, ya solo bastó una función y un diccionario. De igual manera, `os` nos permitió comprobar si el archivo histórico estaba vacío para así darle la estructura adecuada.
3. Tercero, comprendimos la importancia de conocer cómo manipular los datos de un diccionario o lista. En código, cuando queremos usar la base de datos, lo hacemos a través de un diccionario. El proyecto demandaba saber cómo acceder a cada uno de sus valores, ya que debíamos de almacenar y mostrar los adecuados al usuario.
4. Finalmente, entendimos la relevancia del almacenamiento y formato correcto de datos. Durante el trabajo tuvimos que verificar que todo se guarde de manera íntegra y que estos se mantengan en el tiempo. Un simple error o cambio involuntario, haría que el código retorna un resultado no esperado y sería perjudicial para el juego. Asimismo, debíamos de respetar la estructura `JSON` y `.csv`, ya que si agregamos los datos con otro formato, la ejecución fallaría.

Recomendaciones

1. A comparación del trabajo anterior, en esta oportunidad nos resultó más sencillo organizarnos. También, todos estuvieron en las reuniones y si tenían alguna duda preguntaban. Consideramos que debemos mantener esta actitud de mejora en próximos proyectos, para así presentar un buen trabajo.
2. Por otra parte, creemos que hubiera sido adecuado que todos investiguen sobre las nuevas librerías. Como nos dividimos el código, algunos no tuvieron que usarlas, pero a la hora de implementar o comprender el código se les dificultaba. Si bien es cierto, como equipo nos ayudamos cuando alguien no entiende algo, hubiera sido más eficiente si cada uno buscaba el propósito de las funciones importadas.
3. En esta ocasión, desde el primer hasta el último día, todos mostraron interés en el trabajo. Jugaron y experimentaron con el código, y pensamos que al igual que la recomendación 1, debemos de continuar así. Ya que de esta manera podemos conocer más sobre el juego y hacer una especie de testing con cada intento que hacemos.
4. Por último, esta vez empezamos a programar desde antes, sin embargo, tuvimos problemas con el tiempo. A pesar de iniciar ni bien teníamos conocimiento del proyecto, por cuestiones de exámenes en otras asignaturas, se nos resultó complicado avanzar con el informe. Creemos que podríamos solucionar esto mediante una mejor organización en futuras ocasiones o elaborando el informe en paralelo a la programación.