Sudoku (en japonés: 数独, sūdoku) es un juego matemático que se inventó a finales de la década de 1970, adquirió popularidad en Japón en la década de 1984 y se dio a conocer en el ámbito internacional en 2005 cuando numerosos periódicos empezaron a publicarlo en su sección de pasatiempos.

Vamos a desarrollar una aplicación que permite jugar este popular juego. Si no lo conoce, puede leer el artículo de Wikipedia, del cual extraigo acá lo más importante:

El sudoku se presenta normalmente como una tabla de 9 × 9, compuesta por subtablas de 3 × 3 denominadas "**regiones**".

Algunas celdas ya contienen números, conocidos como "números dados" (o a veces "pistas"). El objetivo es rellenar las celdas vacías, con un número en cada una de ellas, de tal forma que cada **columna**, **fila** y **región** contenga los números 1-9 solo una vez.

Además, cada número de la solución aparece solo una vez en cada una de las tres "direcciones", de ahí el "los números deben estar solos" que evoca el nombre del juego.

Preparación

1. Cree una nueva carpeta llamada `sudoku` en alguna parte de su computador.

1. Cree un nuevo archivo en spyder llamado **sudoku_funciones.py** y guárdelo en la carpeta que acaba de crear.

1. Descargue en la carpeta los tres archivos que se encuentran junto a este enunciado: **faciles.txt**, **intermedios.txt**, y **dificiles.txt**

Terminología

Para poder implementar el juego de sudoku, vamos a introducir tres conceptos importantes:

Cadena de juego

Es una representación de un juego de sudoku como una cadena de caracteres numéricos. Por ejemplo, este sería la cadena de un juego terminado:

```
juego_resuelto = "417369825632158947958724316825437169791586432348921758289643571573291684164875293"
```

Sin embargo, podemos representar cualquier estado de un juego, remplazando *por* ceros las posiciones en blanco:

```
juego_inicial = "003020600900305001001806400008102900700000008006708200002609500800203009005010300"
```

Independientemente del estado del juego, la cadena debe tener una longitud fija, que en este caso es 81:

```
print(len(juego_resuelto))
```

```
81
```

Esto es porque la versión más popular de sudoku es una matriz de 9 filas *por* 9 columnas.

Matriz de Juego

Esta matriz es una representación intermedia que vamos a manejar en nuestro programa.

$$S_{9\times9} = \begin{bmatrix} 4 & 1 & 7 & 3 & 6 & 9 & 8 & 2 & 5 \\ 6 & 3 & 2 & 1 & 5 & 8 & 9 & 4 & 7 \\ 9 & 5 & 8 & 7 & 2 & 4 & 3 & 1 & 6 \\ 8 & 2 & 5 & 4 & 3 & 7 & 1 & 6 & 9 \\ 7 & 9 & 1 & 5 & 8 & 6 & 4 & 3 & 2 \\ 3 & 4 & 6 & 9 & 1 & 2 & 7 & 5 & 8 \\ 2 & 8 & 9 & 6 & 4 & 3 & 5 & 7 & 1 \\ 5 & 7 & 3 & 2 & 9 & 1 & 6 & 8 & 4 \\ 1 & 6 & 4 & 8 & 7 & 5 & 2 & 9 & 3 \end{bmatrix}$$

La matriz nos servirá para saber si un juego ha sido resuelto correctamente.

Tablero de Juego

Es una representación de la *Matriz de Juego* como cadena, para que sea más fácil ver las regiones e imprimir todo el tablero en consola:

```
imprimir_tablero(M,(9,9))
```

```
4 1 7 | 3 6 9 | 8 2 5
6 3 2 | 1 5 8 | 9 4 7
9 5 8 | 7 2 4 | 3 1 6
------+-------+------
8 2 5 | 4 3 7 | 1 6 9
7 9 1 | 5 8 6 | 4 3 2
3 4 6 | 9 1 2 | 7 5 8
------+-------+------
2 8 9 | 6 4 3 | 5 7 1
5 7 3 | 2 9 1 | 6 8 4
1 6 4 | 8 7 5 | 2 9 3
```

Desarrollo

Antes de comenzar, vamos a hacer una función de ejemplo para entender cómo armamos la matriz.

1. Copie la siguiente función en su archivo **sudoku.py**, lea el código, y procure entender qué hace la función:

```
def dar_juego_por_filas(cadena:str):
    tamanio = len(cadena)
    raiz = int(math.sqrt(tamanio))
    filas,columnas = raiz,raiz
    k = 0
    juego = ""
    for i in range(filas):
        linea = ""
        for j in range(columnas):
            linea += cadena[k]
            k += 1
        juego += linea + "\n"
    return juego
```

1. Luego, imprima el resultado de la función, usando como **cadena de juego** cualquier juego válido:

```
juego_inicial = "003020600900305001001806400008102900700000008006708200002609500800203009005010300"
print(dar_juego_por_filas(juego_inicial))
```

```
003020600
900305001
001806400
008102900
700000008
006708200
002609500
800203009
005010300
```

Ahora vamos a construir una matriz con cada cadena de juego. Inspírese en la función **dar_juego_por_filas** que acaba de copiar.

1. Desarrolle la siguiente función en el mismo archivo, que retorna una **matriz de juego** a partir de una **cadena de juego** y una tupla de dimensiones:

```
def convertir_a_matriz(juego:str,dimensiones:tuple)->list:
    matriz = []
    # TODO: completar
    return matriz
```

De esta forma podemos probar la función:

```
M = convertir_a_matriz(juego_inicial,(9,9))
print(M)
```

```
[[0, 0, 3, 0, 2, 0, 6, 0, 0], [9, 0, 0, 3, 0, 5, 0, 0, 1], [0, 0, 1, 8, 0, 6, 4, 0, 0], [0, 0, 8, 1, 0, 2, 9, 0, 0], [7, 0, 0, 0, 0, 0, 0, 0, 8], [0, 0, 6, 7, 0, 8, 2, 0, 0], [0, 0, 2, 6, 0, 9, 5, 0, 0], [8, 0, 0, 2, 0, 3, 0, 0, 9], [0, 0, 5, 0, 1, 0, 3, 0, 0]]
```

Que si la pintamos bonito, se veía de la siguiente forma:

$$S_{9\times9} = \begin{bmatrix} 0 & 0 & 3 & 0 & 2 & 0 & 6 & 0 & 0 \\ 9 & 0 & 0 & 3 & 0 & 5 & 0 & 0 & 1 \\ 0 & 0 & 1 & 8 & 0 & 6 & 4 & 0 & 0 \\ 0 & 0 & 8 & 1 & 0 & 2 & 9 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 6 & 7 & 0 & 8 & 2 & 0 & 0 \\ 0 & 0 & 2 & 6 & 0 & 9 & 5 & 0 & 0 \\ 8 & 0 & 0 & 2 & 0 & 3 & 0 & 0 & 9 \\ 0 & 0 & 5 & 0 & 1 & 0 & 3 & 0 & 0 \end{bmatrix}$$

1. Construya la función que imprime el **tablero de juego**:

```
def imprimir_tablero(matriz:list,dimensiones:tuple)->None:
    tablero = ""
    # TODO: completar
    print(tablero)
```

Este sería el resultado esperado:

```
imprimir_tablero(M,(9,9))
```

```
0 0 3 | 0 2 0 | 6 0 0
9 0 0 | 3 0 5 | 0 0 1
0 0 1 | 8 0 6 | 4 0 0
------+-------+------
0 0 8 | 1 0 2 | 9 0 0
7 0 0 | 0 0 0 | 0 0 8
0 0 6 | 7 0 8 | 2 0 0
------+-------+------
0 0 2 | 6 0 9 | 5 0 0
8 0 0 | 2 0 3 | 0 0 9
0 0 5 | 0 1 0 | 3 0 0
```

1. Construya una función que retorne un diccionario de dificultades:

```
def dar_diccionario_juegos()->dict:
    dificultades = ["faciles","intermedios","dificiles"]
    juegos_por_dificultad = {}

    for d in dificultades:
        archivo = open(d+".txt","r")
        # TODO: Completar acá

        archivo.close()
    return juegos_por_dificultad
```

Este diccionario tiene como llave una dificultad, y como valor una lista de **cadenas de juego** que corresponden a esa dificultad. Para ello, procese los tres archivos que están adjuntos al enunciado.

Para probarla, podemos imprimir los primeros 2 sudokus difíciles:

```
lista_dificiles = dar_diccionario_juegos()["dificiles"]
dimensiones = (9,9)
for cadena in lista_dificiles[0:2]:
    M = convertir_a_matriz(cadena,dimensiones)
    imprimir_tablero(M,dimensiones)
```

```
8 5 0 | 0 0 2 | 4 0 0
7 2 0 | 0 0 0 | 0 0 9
0 0 4 | 0 0 0 | 0 0 0
------+-------+------
0 0 0 | 1 0 7 | 0 0 2
3 6 5 | 0 0 0 | 9 0 0
0 4 0 | 0 0 0 | 0 0 0
------+-------+------
0 0 0 | 0 8 0 | 0 7 0
0 1 7 | 0 0 0 | 0 0 0
0 0 0 | 0 3 6 | 0 4 0
------+-------+------
0 0 5 | 3 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 2 0
0 7 0 | 0 1 0 | 5 0 0
------+-------+------
4 0 0 | 0 0 5 | 3 0 0
0 1 9 | 0 7 0 | 0 6 0
0 0 3 | 2 0 0 | 0 0 8
------+-------+------
0 0 9 | 3 0 0 | 0 6 0
0 0 4 | 0 0 0 | 0 3 0
0 0 0 | 0 0 9 | 7 0 0
```

1. Cree una función que me retorne la **matriz de juego** de un sudoku escogido al azar:

```
def dar_juego_aleatorio_por_dificultad(dificultad:str)->str:
    # TODO: Completar
```

Puede usar la función que crea el diccionario de juegos.

```
juego = dar_juego_aleatorio_por_dificultad("intermedios")

dimensiones = (9,9)
M = convertir_a_matriz(juego,dimensiones)
imprimir_tablero(M,dimensiones)
```

```
0 0 0 | 5 0 0 | 0 0 0
0 9 0 | 4 0 1 | 5 8 6
0 7 0 | 0 0 0 | 9 2 0
------+-------+------
0 0 4 | 1 0 2 | 0 0 0
2 5 0 | 1 9 0 | 0 3 0
0 0 0 | 0 3 0 | 0 0 0
------+-------+------
0 0 0 | 0 0 4 | 0 7 0
0 1 3 | 0 0 0 | 0 0 0
0 0 0 | 2 0 0 | 0 0 0
```

Verificando un juego

Vamos a tener una función que verifica si un juego es correcto. Copie la siguiente función en su archivo:

```
def verificar_juego(matriz:list)->bool:
    return verificar_filas(matriz) and verificar_columnas(matriz) and verificar_regiones(matriz)
```

1. Desarrolle la siguiente función, que verifica si todas las filas de la matriz de juego son correctas:

```
def verificar_filas(matriz: list) -> bool:
    correcto = True
    # TODO: Completar
    return correcto
```

1. Desarrolle la siguiente función, que verifica si todas las columnas de la matriz de juego son correctas:

```
def verificar_columnas(matriz:list)->bool:
    correcto = True
    # TODO: Completar
    return correcto
```

1. Desarrolle la siguiente función, que verifica si todas las regiones de la matriz de juego son correctas:

```
def verificar_regiones(matriz:list)->bool:
    correcto  = True
    # TODO: Completar
    return correcto
```

Podemos probar la verificación de la siguiente forma:

```
dimensiones = (9,9)
M = convertir_a_matriz(juego_inicial,dimensiones)
imprimir_tablero(M,dimensiones)
print("El juego es correcto?",verificar_juego(M))
```

```
0 0 3 | 0 2 0 | 6 0 0
9 0 0 | 3 0 5 | 0 0 1
0 0 1 | 8 0 6 | 4 0 0
------+-------+------
0 0 8 | 1 0 2 | 9 0 0
7 0 0 | 0 0 0 | 0 0 8
0 0 6 | 7 0 8 | 2 0 0
------+-------+------
0 0 2 | 6 0 9 | 5 0 0
8 0 0 | 2 0 3 | 0 0 9
0 0 5 | 0 1 0 | 3 0 0
El juego es correcto? False
```

Este sería el resultado para un juego resuelto correctamente:

```
juego_resuelto = "417369825632158947958724316825437169791586432348921758289643571573291684164875293"

dimensiones = (9,9)
M = convertir_a_matriz(juego_resuelto,dimensiones)
imprimir_tablero(M,dimensiones)
print("El juego es correcto?",verificar_juego(M))
```

```
4 1 7 | 3 6 9 | 8 2 5
6 3 2 | 1 5 8 | 9 4 7
9 5 8 | 7 2 4 | 3 1 6
------+-------+------
8 2 5 | 4 3 7 | 1 6 9
7 9 1 | 5 8 6 | 4 3 2
3 4 6 | 9 1 2 | 7 5 8
------+-------+------
2 8 9 | 6 4 3 | 5 7 1
5 7 3 | 2 9 1 | 6 8 4
1 6 4 | 8 7 5 | 2 9 3
El juego es correcto? True
```

Incluso podríamos generar un juego terminado, pero incorrecto, si queremos probar a fondo:

```
juego_incorrecto = ""
digitos = "123456789"
for i in range(82):
    juego_incorrecto += digitos[random.randint(0,len(digitos)-1)]

dimensiones = (9,9)
M = convertir_a_matriz(juego_incorrecto,dimensiones)
imprimir_tablero(M,dimensiones)
print("El juego es correcto?",verificar_juego(M))
```

```
7 5 7 | 6 5 6 | 4 7 6
7 9 5 | 1 4 9 | 1 1 4
7 3 2 | 1 7 5 | 9 9 8
------+-------+------
8 7 1 | 9 4 6 | 4 7 4
1 7 1 | 6 7 9 | 6 2 3
1 5 6 | 2 2 6 | 8 3 5
------+-------+------
6 3 3 | 6 1 3 | 8 2 5
9 8 3 | 3 4 9 | 2 3 3
5 9 1 | 4 2 6 | 7 7 7
El juego es correcto? False
```

Entrega

Suba el archivo **sudoku_funciones.py** a Brightspace en la actividad correspondiente.