



Práctica 5: Programación modular usando funciones

1. Objetivos

- Comprender los conceptos básicos sobre la codificación de funciones en Python.
- Desarrollar programas que involucren programación modular mediante el uso de funciones.
- Aprender a manipular archivos para los datos de entrada y salida de un programa.

2. Marco teórico

Las funciones, presentes en la gran mayoría de lenguajes de programación, son la manera como un lenguaje le permite al programador crear sub-programas. Con las funciones, el programador puede encapsular un segmento de código que en la mayoría de los casos va a ser utilizado múltiples veces, o simplemente, le permite organizar un programa de manera modular. Al crear una función se debe dejar claro, por medio de un docstring, la especificación que describe las condiciones de los argumentos (datos de entrada) y las garantías del trabajo que hace la función. Las funciones deben ser probadas, una por una, a medida que se van integrando al programa principal.

3. Tareas a realizar

Esta práctica consiste en la implementación de un programa para el juego clásico de **triqui**. El programa debe ser desarrollado de manera modular mediante el uso de funciones.

Resumen del juego

Triqui se jugará entre el usuario y el computador, donde uno de los jugadores usa la marca "X" y el otro la marca "O". Los jugadores toman turnos para ubicar sus marcas en el tablero. Si un jugador logra posicionar tres de sus marcas en una fila, una columna o en una diagonal del tablero este ganará el juego. Si el tablero se llena sin que ninguno de los jugadores gane, se declarará un empate. En la siguiente página: <https://playtictactoe.org> puede experimentar con el juego para que comprenda su funcionamiento.

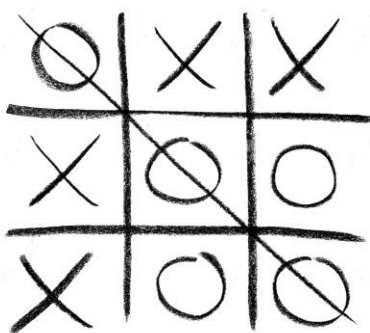


Figura 1. Tablero de triqui.

Funcionamiento del programa

El usuario jugará contra el computador, y se debe implementar un algoritmo simple que le permita al computador responder de forma coherente a las jugadas del usuario. A continuación se dan los requerimientos que deberá cumplir el programa a implementar:

- Cuando el programa inicia por primera vez deberá tener un anuncio de bienvenida con el nombre del juego de manera semi-gráfica.
- Una vez se inicie el juego, éste deberá permitir seleccionar la marca que usará el usuario ("X" o "O").
- Luego el programa debe decidir de forma aleatoria quién inicia el juego.
- El usuario y la computadora deben tomar turnos para ubicar sus marcas en el tablero. Si el usuario intenta ubicar una marca en un lugar ya ocupado por otra marca, el programa deberá solicitarle al usuario que ingrese su marca de nuevo.
- En cada turno, el programa deberá mostrar el estado actual del tablero, con las posiciones ocupadas por las marcas del usuario y el computador.
- El juego culmina cuando el usuario o el computador logra posicionar tres de sus marcas en una fila, una columna o en una diagonal del tablero. Si el tablero se llena sin que ninguno de los jugadores gane, se declarará un empate.
- Cuando el juego termina, deberá preguntar al usuario si quiere empezar un nuevo juego o si prefiere salir del programa.

Restricciones adicionales del juego

- El programa debe restringir los datos que el usuario puede ingresar, datos inválidos deben ser ignorados por el programa, y se le debe solicitar al usuario que ingrese datos válidos.

Diagrama de flujo principal

Este problema es un poco más complejo que los que se han venido desarrollando hasta el momento, por lo tanto lo ideal antes de codificarlo es planificar su estructura de funcionamiento general. Una herramienta bastante útil para ello consiste en el diagrama de flujo, que en nuestro caso, serán diagramas de flujos simplificados y abstractos pues el detalle debe ser desarrollado por el estudiante. Iniciaremos describiendo el diagrama de flujo general del juego, y luego, una vez definido este, se definirán todos aquellos detalles internos necesarios para la implementación de cada proceso del diagrama de flujo. Aunque el desarrollo del diagrama de flujo no suele ser algo obligatorio cuando se concibe un programa, recomendamos que se tome su tiempo para esto y no se apresure con iniciar la codificación. A continuación la figura 2 muestra el diagrama de flujo general del programa.

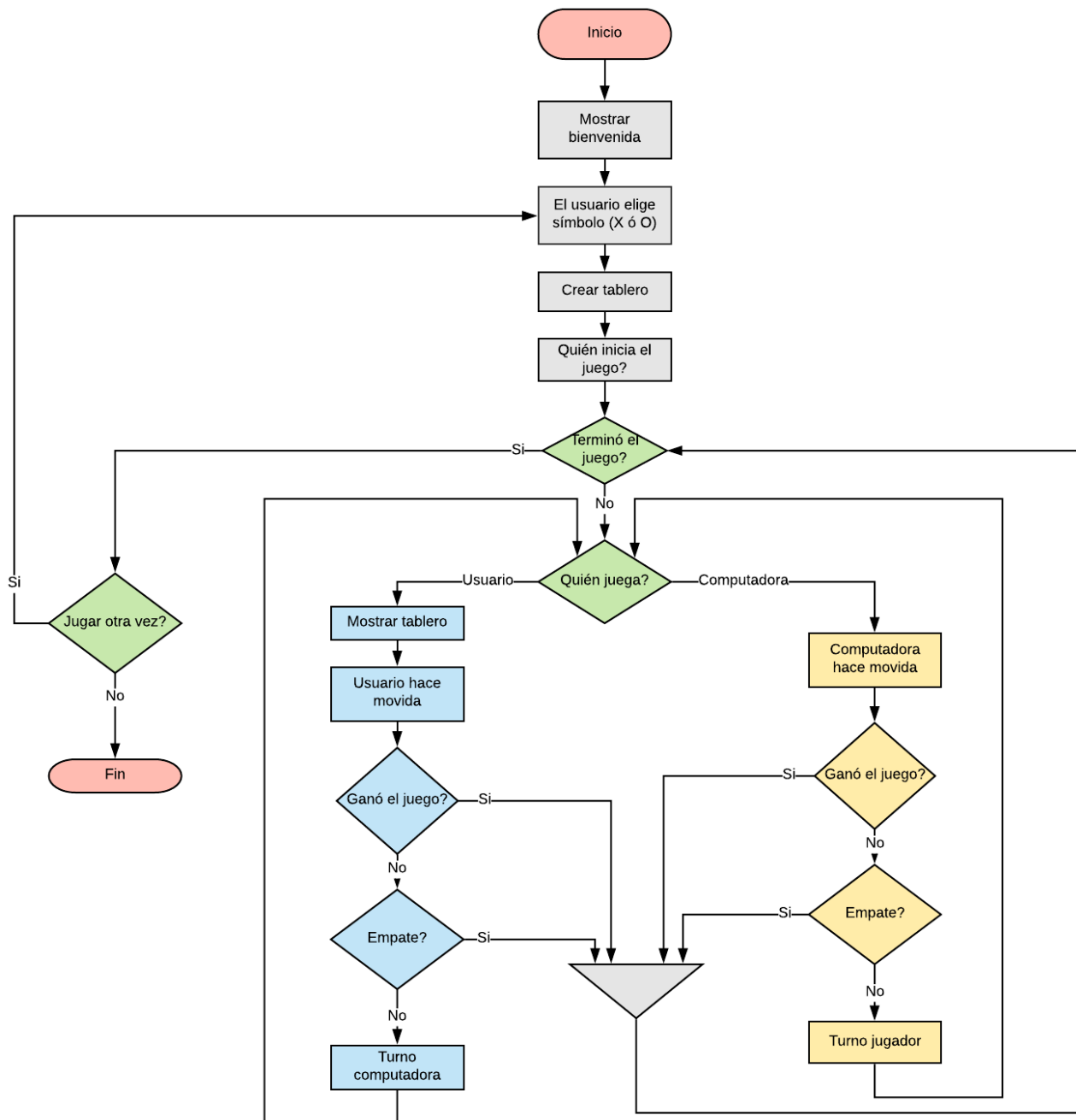


Figura 2. Diagrama de flujo principal del juego.

La figura anterior muestra el funcionamiento del programa de modo resumido. El propósito de este laboratorio es que usted implemente esta funcionalidad haciendo uso de funciones.

Desarrollo del programa

Para el desarrollo de este laboratorio se suministran los siguientes archivos los cuales deberán ser descargados de la página del curso:

- **triqui.py**: Archivo donde se hará la implementación de todas las funciones necesarias para el desarrollo del juego triqui.

- **main.py**: Archivo en el que se implementará el programa principal haciendo uso de las funciones implementadas en el archivo `triqui.py`.
- **intro.txt**: Archivo que contiene el nombre del juego de manera amigable.

Anuncio de bienvenida

El anuncio a desplegar se encuentra almacenado en el archivo **intro.txt**. La función llamada **printIntro()** deberá desarrollarse (en el archivo `triqui.py`) para permitir desplegar el contenido de cualquier archivo de texto cuyo nombre es tomado como argumento de entrada. Pruebe la función con el archivo **intro.txt** suministrado.

Una vez la función **printIntro()** esté completa, el siguiente paso consistirá en probar que funcione bien. En los comentarios de la función en la plantilla (en la sección de ejemplos de uso), se muestra la salida tentativa una vez se hace el llamado a la función. Si todo está bien, la salida deberá ser algo como se muestra en la figura 3.

```
>>> printIntro("intro.txt")
```

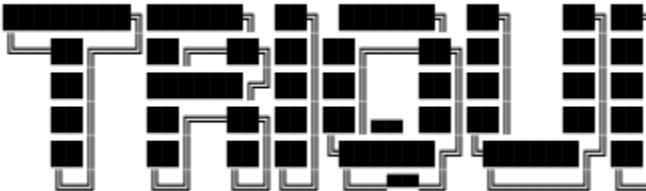


Figura 3. Resultado de la invocación de la función `printIntro()`.

Comentarios y tips:

- Recuerde el tema de manipulación de archivos:
<http://docs.python.org.ar/tutorial/3/inputoutput.html#leyendo-y-escribiendo-archivos>
- Tenga en cuenta que el archivo **intro.txt** le será entregado y no será necesario que lo modifique o cree nuevos archivos para desplegar la entrada semi-gráfica. Sin embargo, si desea experimentar, consulte la el sitio web:
<http://patorik.com/software/taag/#p=display&f=ANSI%20Shadow&t=triqui>
- El archivo `intro.txt` está formateado para Linux, por lo tanto si realiza esta práctica en Windows, es posible que el archivo sea interpretado de forma incorrecta y Python generará errores al intentar abrirlo.

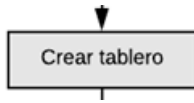
Selección del símbolo por parte del usuario



El usuario debe poder elegir entre las dos marcas características del juego triqui, esto es, el usuario debe escoger la marca "X" o la marca "O". Para implementar esta parte complete la función **inputPlayerLetter()** (en `triqui.py`) la cual se encarga de solicitar al usuario que elija entre la letra "X" y la letra "O". Esta función no tendrá argumentos de entrada y deberá retornar una lista con la letra

elegida por el usuario en la primera posición y la letra usada por el computador en la segunda posición. Si el usuario intenta elegir una letra o carácter distinto de "X" o "O", el programa deberá solicitarle al usuario que elija nuevamente.

Crear tablero



Primero que todo se debe decidir cómo se va a representar el tablero en el programa. Normalmente, el tablero es representado como un par de líneas horizontales entre cruzadas con un par de líneas verticales, con "X", "O", o espacios vacíos en las nueve casillas del tablero (ver figura 1). Se debe representar a las nueve casillas del tablero como una lista de strings, donde cada string puede ser "X", "O", o un espacio (" "). Para recordar fácilmente que índice de la lista de strings corresponde a una casilla del tablero, usaremos la misma numeración que encontramos en el teclado numérico de una computadora (ver figura 4).

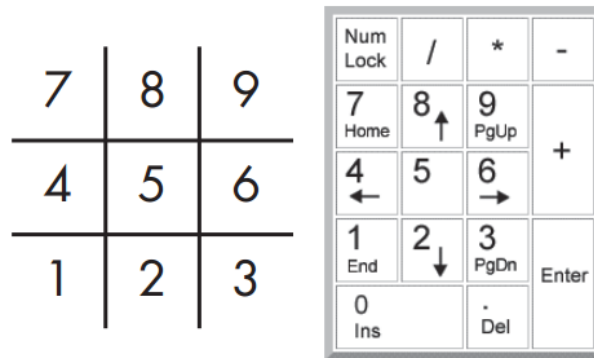


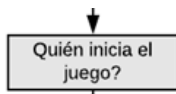
Figura 4. Las casillas del tablero se numeran como en un teclado numérico.

Así, si se usa una lista de diez strings para representar al tablero, entonces lista[7] correspondería a la casilla superior izquierda del tablero, lista[9], correspondería a la casilla superior derecha del tablero, lista[5] correspondería a la casilla central del tablero, etc. Lista[0] no sería usada en el programa ya que no representa ninguna casilla del tablero. Al inicio de cada partida se debe crear el tablero con todas sus casillas vacías.

Comentarios y tips:

- Recuerde el tema de manipulación de listas en Python:
<https://docs.python.org/3.6/tutorial/introduction.html#lists>

Decidir quién inicia el juego

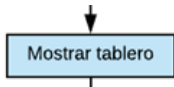


El programa debe decidir de forma aleatoria si el usuario toma el primer turno o el computador toma el primer turno. La función **whoGoesFirst()** (en triqui.py) debe implementar esta funcionalidad. Esta función debe retornar el string "Computadora" para indicar que el computador va primero, y debe retornar el string "Usuario" para indicar que el usuario va primero.

Comentarios y tips:

- Para implementar **whoGoesFirst()** le puede servir alguna de las funciones del modulo random: <https://docs.python.org/3/library/random.html>

Mostrar tablero



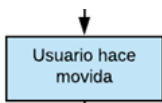
Cada vez que el usuario tenga el turno para hacer una jugada, se deberá presentar el estado actual del tablero. La función **drawBoard(board)** (en triqui.py) se encargará de implementar esta funcionalidad. Esta función aceptará como argumento la lista de strings (board) que almacena el estado del tablero (ver **crear tablero**), y deberá imprimir en la consola el estado actual del tablero. En la figura 5 se muestra dos ejemplos del uso de esta función.

```
>>> drawBoard([' ', ' ', ' ', ' ', ' ', 'X', 'O', ' ', ' ', 'X', ' ', ' ', 'O'])
X| |O
--+--+
X|O|
--+--+
| |
>>> drawBoard([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '])
| |
--+--+
| |
--+--+
| |
```

Figura 5. Ejemplos de uso de la función drawBoard()

El programa debe posicionar cada string de la lista en la casilla correspondiente de acuerdo a la numeración presentada en la figura 4.

Usuario hace movida



Cuando es turno del usuario, este debe estar en capacidad de hacer una movida en el tablero. Esta funcionalidad es un poco más compleja y por tanto se implementará usando varias funciones:

- **getPlayerMove(board):** Esta función le pide al usuario que ingrese el número de la casilla en la cual quiere posicionar su marca (1 a 9). Esta función acepta como argumento la lista de strings que representa el estado actual del tablero (board). Esta función debe verificar que el usuario ingrese un número de casilla válido y que esta casilla está vacía. Para este último, debe usar la función **isSpaceFree()** la cual se definirá a continuación.

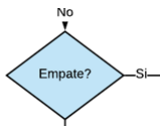
- **isSpaceFree(board,move):** Esta función debe verificar que la casilla seleccionada por el usuario se encuentra vacía. Esta función toma como argumentos la lista de strings que representa el estado actual del tablero (board) y la variable move que almacena el número de la casilla seleccionada por el usuario. Esta función debe retornar el valor lógico **True** si la casilla está vacía y el valor lógico **False** si la casilla está ocupada.
- **makeMove(board,letter,move):** Esta función debe actualizar el estado del tablero, poniendo en la posición indicada por la variable move, la letra ("X" o "O") almacenada en la variable letter.

Verificar si hay un ganador



Después de que se haya puesto una nueva marca en el tablero, es necesario verificar si hay un ganador. La función **isWinner(board,letter)** (en triqui.py) debe implementar esta funcionalidad. Esta función tomará como argumentos la lista de strings que representa el estado del tablero (board) y la variable que almacena la letra que está usando el usuario o el computador ("X" o "O"). Hay ocho formas de ganar en el juego de triqui, se puede tener 3 marcas en cualquiera de las tres filas del tablero, se puede tener 3 marcas en cualquiera de las tres columnas del tablero, o tres marcas en cualquiera de las dos diagonales del tablero. Esta función deberá verificar si alguna de estas condiciones se cumple, y deberá retornar el valor lógico **True** en caso de que haya una jugada ganadora, o en su defecto, deberá retornar el valor lógico **False**.

Verificar si hay un empate

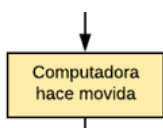


Un empate se da cuando todas las casillas del tablero están ocupadas, pero no hay ninguna jugada ganadora. La función **isBoardFull(board)** deberá implementar esta verificación. Esta función acepta como argumento la lista de strings que representa el estado del tablero (board) y deberá retornar el valor lógico **True** si hay un empate, o en su defecto deberá retornar el valor lógico **False**.

Comentarios y tips:

- Para la implementación de **isBoardFull()** es necesario utilizar la función **isSpaceFree()**, la cual fue implementada anteriormente.

Computadora hace movida



La estrategia de juego del computador consistirá en un simple algoritmo, el cual se muestra en la figura 6.

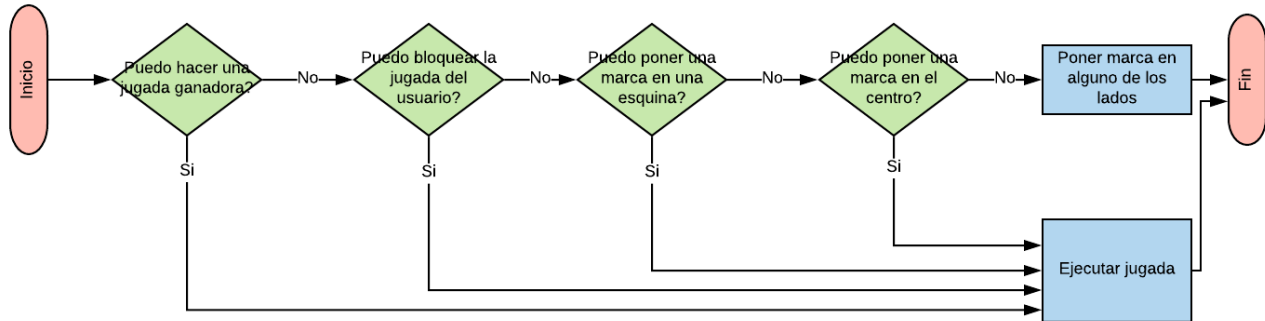


Figura 6. Estrategia de juego de el computador.

La computadora debe analizar el estado actual del juego y decidir cuál será su próxima movida. Para mayor claridad se indicarán tres tipos de casillas en el tablero: esquinas, lados y centro. La figura 7 muestra esta nomenclatura.

Esquina (7)	Lado (8)	Esquina (9)
Lado (4)	Centro (5)	Lado (6)
Esquina (1)	Lado (2)	Esquina (3)

Figura 7. Nomenclatura casillas del tablero.

Según la figura 6, la estrategia de juego del computador consiste en los siguientes pasos:

1. Verificar si hay alguna movida que el computador pueda hacer, que haga que este gane el juego. Si la hay, ejecutar la jugada. En su defecto, ir al paso 2.
2. Verificar si hay alguna movida que el usuario pueda hacer, que haga que este gane el juego. Si la hay, bloquear esta movida. En su defecto, ir al paso 3.
3. Verificar si alguna de las esquinas del tablero están vacías (casillas 1, 3, 7 y 9). Si alguna está vacía, marcar esta casilla. Si ninguna de las casillas de las esquinas esta vacía, ir al paso 4.
4. Verificar si la casilla del centro (casilla 5) está vacía. Si lo está, marcar esta casilla. Si la casilla del centro no está vacía, ir al paso 5.
5. Marcar alguna de las casillas de los lados que esté vacía (casillas 2, 4, 6, 8).

La función **getComputerMove(Board,computerLetter)** (en triqui.py) debe implementar esta funcionalidad. Esta función toma como argumento la lista de strings que representa el estado del tablero (board) y la variable computerLetter que almacena la marca usada por el computador ("X" o "O"). Esta función debe retornar un número entero entre 1 y 9, que indica la casilla a marcar. A continuación se proporcionan más detalles sobre la funcionalidad a implementar.

Verificar si el computador puede ganar en la siguiente jugada

Antes que todo lo demás, si existe una jugada con la cual el computador puede ganar, el computador debe ejecutar esta jugada inmediatamente. Así, se debe verificar que si al marcar alguna de las casillas disponibles actualmente en el tablero, el computador obtendría alguna de las ocho formas de ganar (ver **verificar si hay un ganador**).

Comentarios y tips:

- Para la implementación de esta funcionalidad es necesario utilizar las funciones **isSpaceFree()**, **makeMove()** y **isWinner()**, las cuales fueron implementadas anteriormente.
- También se recomienda crear una copia de la lista de strings que almacena el estado del tablero, e implementar esta funcionalidad usando esta copia, para no poner en peligro la integridad de la lista de strings original. Información sobre cómo hacer copias de una lista y sus implicaciones se encuentra en el siguiente link: <https://docs.python.org/3.6/library/copy.html>

Verificar si el usuario puede ganar en la siguiente jugada

Si existe una jugada con la cual el usuario puede ganar el juego, el computador debe bloquear esta jugada inmediatamente. Así, se debe verificar que si al marcar alguna de las casillas disponibles actualmente en el tablero, el usuario obtendría alguna de las ocho formas de ganar (ver **verificar si hay un ganador**).

Comentarios y tips:

- Para la implementación de esta funcionalidad es necesario utilizar las funciones **isSpaceFree()**, **makeMove()** y **isWinner()**, las cuales fueron implementadas anteriormente.
- También se recomienda crear una copia de la lista de strings que almacena el estado del tablero, e implementar esta funcionalidad usando esta copia, para no poner en peligro la integridad de la lista de strings original. Información sobre cómo hacer copias de una lista y sus implicaciones se encuentra en el siguiente link: <https://docs.python.org/3.6/library/copy.html>

Verificar si alguna de las casillas de las esquinas, del centro, o los lados está vacía

Si el computador no puede hacer una movida ganadora, y no necesita bloquear una posible jugada ganadora del usuario, lo siguiente que puede hacer es poner una marca en alguna de las casillas de las esquinas, el centro, o los lados. La función **chooseRandomMoveFromList(board, moveList)** le debe servir para implementar esta funcionalidad. Esta función toma como argumentos la lista de strings que almacena el estado actual del tablero (board), y una lista que tiene como elementos los números de las casillas que se quieren evaluar (moveList). Esta función debe retornar de forma aleatoria algún elemento de la lista moveList, si este elemento representa una casilla vacía. Si ninguna de las casillas está vacía, esta función debe retornar **None**. Por ejemplo, si se desea verificar si alguna de las casillas de las esquinas está vacía, moveList = [1,3,7,9], si se desea verificar si alguna de las casillas de los lados está vacía, moveList = [2,4,6,8]. Para poner una marca en la casilla central del tablero, solo se debe verificar que esta casilla está vacía.

Comentarios y tips:

- Para implementar **chooseRandomMoveFromList ()** le puede servir alguna de las funciones del módulo random: <https://docs.python.org/3/library/random.html>

- Para implementar esta función debe usar la función **isSpaceFree()**, la cual fue implementada anteriormente.

Integración de las funciones para implementar el juego completo

Antes de iniciar esta fase de desarrollo, asegúrese de haber verificado el correcto funcionamiento de cada una de las funciones. Ahora lo que resta es integrar las funciones en el programa que implementará el juego como tal. Para esto se proporcionará el esqueleto del archivo principal (main.py). Sin embargo describiremos antes algunas variables importantes que le serán dadas y que deberá emplear en la implementación del juego; estas son:

- **turn:** Esta variable indica quién tiene el turno para jugar, el usuario o el computador.
- **jugando:** Esta variable indica si el juego está en marcha (**True**) o debe terminar (**False**).

Guíese por el código y los comentarios que hay en la plantilla del programa principal y el diagrama de flujo de la figura 2.

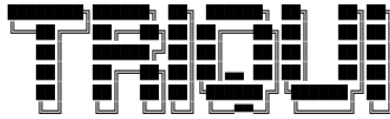
Opcional

Agregue una función que al finalizar el programa permita guardar en un archivo las estadísticas del juego, es decir, cuantas veces ha ganado el usuario, cuantas veces a ganado el computador, y cuantos empates se han presentado. El formato del archivo se muestra en la figura 8.

```
----- Triqui -----  
Tue Aug 28 17:50:32 -05 2018  
Nombre del jugador: Juanito  
Juegos ganados por el usuario = 1  
Juegos ganados por el computador = 5  
Juegos empatados = 3  
-----  
----- Triqui -----  
Tue Aug 28 18:10:15 -05 2018  
Nombre del jugador: Pepito  
Juegos ganados por el usuario = 0  
Juegos ganados por el computador = 3  
Juegos empatados = 1  
-----
```

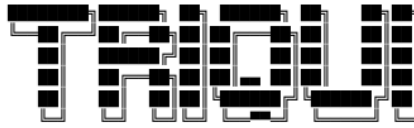
Figura 8. Formato del archivo de estadísticas del juego.

A continuación se muestran ejemplos de varias sesiones del programa ante diferentes situaciones. Obsérvelas bien pues le ayudaran a implementar el programa según los requerimientos.



```
Elige entre X o O
X
Usuario va primero.
| |
-+-+
| |
-+-+
| |
¿Cual es tu siguiente jugada? (1-9)
9
O| |X
-+-+
| |
-+-+
| |
¿Cual es tu siguiente jugada? (1-9)
2
O| |X
-+-+
| |
-+-+
|X|O
¿Cual es tu siguiente jugada? (1-9)
5
O| |X
-+-+
|X|
-+-+
O|X|O
¿Cual es tu siguiente jugada? (1-9)
8
O|X|X
-+-+
|X|
-+-+
O|X|O
Felicitaciones, has ganado el juego!
¿Quieres jugar otra vez? (si o no)
|
```

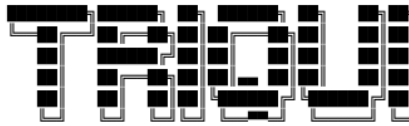
Figura 9. Caso en el que el usuario gana la partida.



```
Elige entre X o O
0
Computadora va primero.
| |X
-+-+
| |
-+-+
| |
¿Cual es tu siguiente jugada? (1-9)
1
X| |X
-+-+
| |
-+-+
O| |
¿Cual es tu siguiente jugada? (1-9)
8
X|O|X
-+-+
| |
-+-+
O| |X
¿Cual es tu siguiente jugada? (1-9)
8
X|O|X
-+-+
| |
-+-+
O| |X
¿Cual es tu siguiente jugada? (1-9)
6
X|O|X
-+-+
|X|O
-+-+
O| |X
La computadora te ha vencido! Perdiste.
¿Quieres jugar otra vez? (si o no)
no

Process finished with exit code 0
|
```

Figura 10. Caso en el que el usuario pierde la partida.



```

Elige entre X o O
x
Computadora va primero.
| |
-+-+
| |
-+-+
O| |
¿Cual es tu siguiente jugada? (1-9)
5
| |O
-+-+
|X|
-+-+
O| |
¿Cual es tu siguiente jugada? (1-9)
6
| |O
-+-+
O|X|X
-+-+
O| |
¿Cual es tu siguiente jugada? (1-9)
7
X| |O
-+-+
O|X|X
-+-+
O| |O
¿Cual es tu siguiente jugada? (1-9)
2
X|O|O
-+-+
O|X|X
-+-+
O|X|O
Juego empatado!
¿Quieres jugar otra vez? (si o no)
no

Process finished with exit code 0
|

```

Figura 11. Caso en el que el juego queda empatado.

4. Evaluación

La evaluación se basará en los códigos enviados y un quiz sobre los temas de la práctica. Además, se tendrá en cuenta una bonificación para quien haga la parte opcional.