Create: noughts and crosses

# Task . Getting started

Create a new Python file in your development environment and copy over the code that you completed in the last paired activity. This will be your starting point for this project.

Remember that you have already completed two sections of the structure chart:



# Task . Game instructions

**Step 1**

A procedure needs to be created to provide instructions to the user at the beginning of the game.

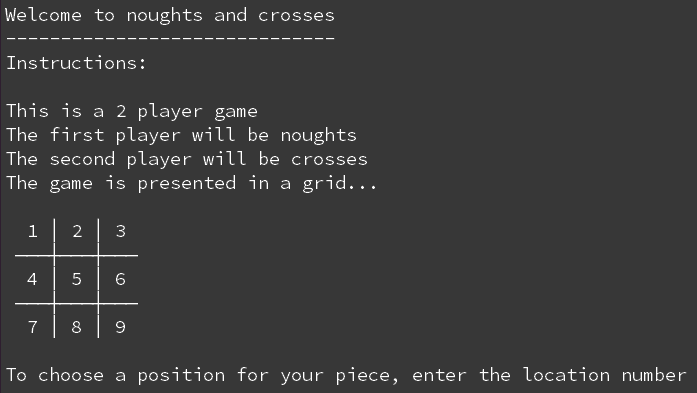
During game play, the user will see the board with the numbers 1 to 9 to show the locations where they can place their X or O. They enter the number and then their X or O will be placed on the board. The first player will be X, and the second player will be O.

**Create a procedure** called instructions that provides these instructions to the user.

**Step 2**

Test your program by calling the procedure to see how user friendly it is. You might decide that it needs to appear more slowly and introduce some delays to the program to slow the instructions down.

An **example output** can be seen below:



# Task . Move a player piece

**Step 1**

A function needs to be created that moves the player piece onto the board. Define a function using the following interface:

Identifier: move

Parameters: board, player

Return values: board

**Tip:** All subroutines should be defined before the main program begins.

**Step 2**

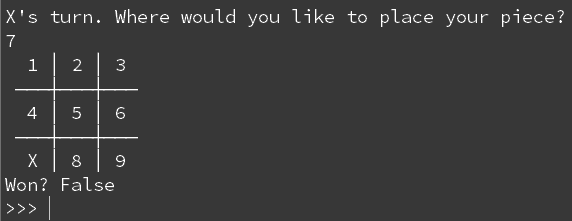
Complete the function ensuring that it:

* Asks the current player to enter their desired position
* Uses the position entered to add either a O or X to the board at the desired location
* Returns the board

**Step 3**

For testing purposes, make sure that player holds the value X. Test your program by entering each number from 1 to 9 and making sure that it correctly adds an X to the correct position on the board.

An **example output** can be seen below:



**Tip:** When testing your code, you can # tag out the instructions() procedure to make testing quicker. Just place a # before the procedure call and it will be ignored during execution.

# Task . Play

**Step 1**

A procedure needs to be created that will run the playing of the game. This will be built up over several tasks. Define a procedure using the following interface:

Identifier: play

Parameters: board

Return values: none

**Step 2**

At this stage, the play procedure should:

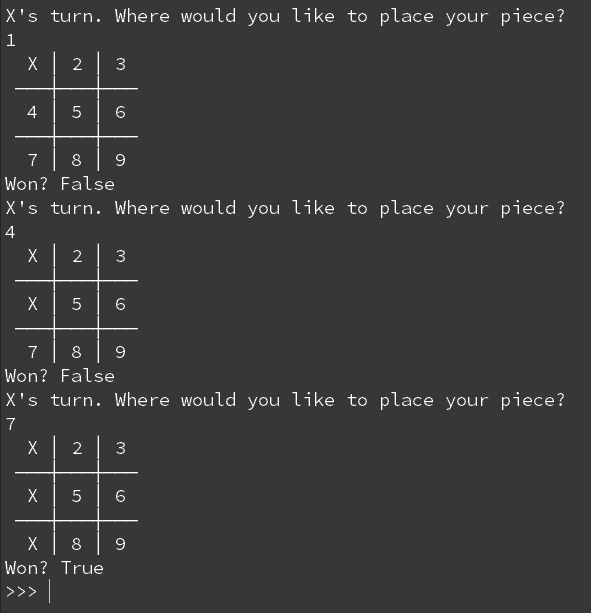
* Only work for X
* Continue to ask the user to enter a position until X wins
* End the program when X wins

Complete the play procedure so that it performs the above tasks.

**Step 3**

Test your program for all winning moves that X can have to make sure that it works correctly. **Remember** to call the procedure at the bottom of your program.

An **example output** can be seen below:

****

# Task . Switching between players

**Step 1**

For this task you will still be working in the play procedure.

Introduce a new variable called currentplayer and hold the value X. This should be declared outside of the while loop initially.

**Step 2**

Introduce selection to your while loop so that if the current player is X it switches to O, and if the current player is O it switches to X.

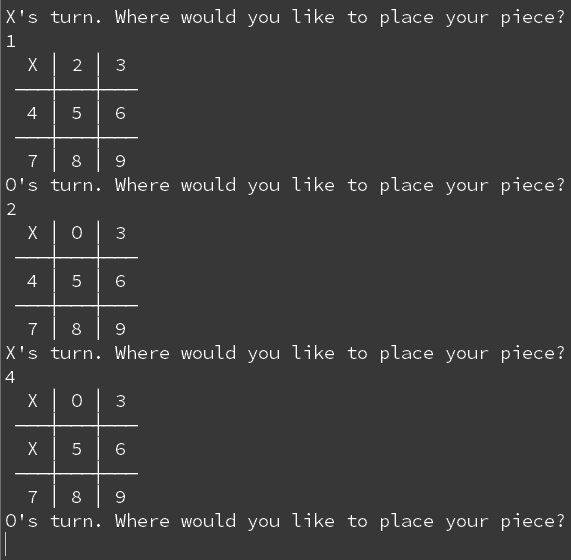
**Step 3**

Ensure that player now holds currentplayer instead of X.

**Step 4**

Test your program to see if it switches between players during game play.

An **example output** can be seen below:



# Task . Announce the winner!

**Step 1**

Introduce a new variable called winner. The new variable should hold the currentplayer when a win happens.

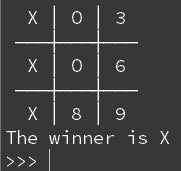
**Step 2**

When a player has won, create an output that states which player has won the game.

**Step 3**

Test the game to make sure that the correct winner is announced at the end of the game.

An **example output** can be seen below:



# Task . Add validation checks

**Step 1**

Create a new function that will make sure that a **string** number between 1 and 9 has been entered by the player in the move function. The function should have the following interface:

Identifier: checkpositions

Parameters: player

Return values: position

**Step 2**

The checkpositions function should:

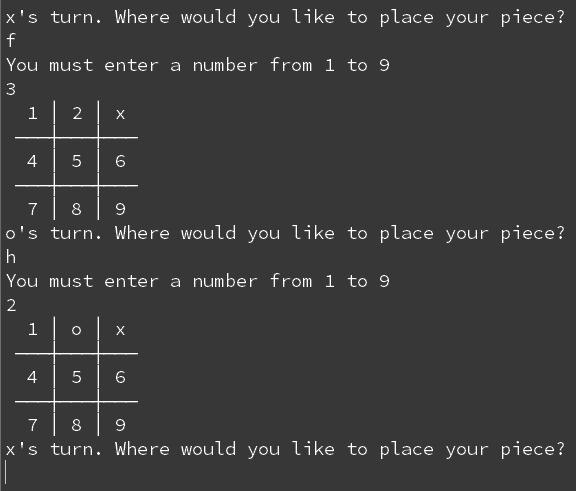
* Ask which location the piece should be placed
* Check that the location is a **string** number between 1 and 9
* If the value is entered incorrectly, then it should continue to ask for the correct input
* Return the valid value back to the move function to the position variable

**Tip:** You need to make sure that the user is entering a string character and not an integer. You can use a list to hold the valid values and check the input against the list.

**Step 3**

Test that your new function works correctly by entering correct and incorrect data.

An **example output** is shown below:



**Step 4**

Ensure that appropriate validation checks have been used at any other input in the game.

# Submit checklist .

Before handing in your code, make sure that you have checked off as many of the sections as you can.

**The program should:**

✔ Use this as a checklist for your program

|  | Provide instructions for the user |
| --- | --- |
|  | Switch between players |
|  | Add the current players piece to the board list |
|  | Display the board with the newly added piece |
|  | Continue to play until a player wins |
|  | Reveal the winner at the end of the game |
|  | Use validation checks for any inputs |

# Explorer tasks .

Improve your program by:

* Checking if the game is a draw and displaying that as an alternative to the winner
* Adding a ‘play again’ feature
* Adding extra functionality that stops a player from overwriting another players piece
* Adding an option to play against the computer
* Recreating the whole program in guizero

Your Final Code:

|  |
| --- |