Finishing the Tic-Tac-Toe Program

by Sophia

WHAT'S COVERED

In this lesson, we will finish our Tic-Tac-Toe program to make use of functions. Specifically, this lesson covers:

1. Improved Board
2. Checking for Winners

1. Improved Board

We have come back to our Tic-Tac-Toe game that we worked on previously and will be optimizing it and making use of functions to help with repeatability. During the prior edits of our Tic-Tac-Toe game, we incorporated some validation and the use of loops.

Here is the code from when we last left the game.

TRY IT

Directions: If you don't have an existing GDB project, let's enter in the following code to have what we originally ended with the last time we visited the Tic-Tac-Toe program.

EXAMPLE

```
board = [ ["-", "-", "-"],
          ["-", "-", "-"],
          ["-", "-", "-"] ]

print(board[0])
print(board[1])
print(board[2])
col=0
row=0
playerTurn = "X"
```

```
for counter in range(1,10):

    validMove = False
    while (validMove == False):
      col=0
      row=0
      while (col < 1 or col > 3):
        col = int(input(playerTurn + " player, select a column 1-3: "))
        if (col < 1 or col > 3):
          print("The column must be between 1 and 3.")
      while (row < 1 or row > 3):
        row = int(input(playerTurn + " player, select a row 1-3: "))
        if (row < 1 or row > 3):
          print("The row must be between 1 and 3.")
      col -= 1
      row -= 1
      if board[row][col] == '-':
        board[row][col] = playerTurn
        validMove=True;
      else:
        print("Oops, that spot was already taken. Please select another spot.")

    print(board[0])
    print(board[1])
    print(board[2])

    if playerTurn =="X":
      playerTurn="O"
    else:
      playerTurn="X"
```

TRY IT

Directions: Go ahead and run the program a few times to get reacquainted with the gameplay.

When we are creating functions, we want to think about what types of functionality may repeat in our program and thus be a good fit for the creation of a function. This way, if there are changes that are needed, we'll only have to make the changes in one place. One place where we can see an immediate need is the output of the board. We currently have that at the start and then again after the player has made their move (those repeat print(board..) lines)).

We can create a function that would quickly change that.

EXAMPLE

```
def printBoard(board):
  print(board[0])
  print(board[1])
  print(board[2])
```

And then in any place where we produced the output of the board, we can call the `printBoard()` function instead.

EXAMPLE

```
board = [ ["-", "-", "-"],
          ["-", "-", "-"],
          ["-", "-", "-"] ]

def printBoard(board):
  print(board[0])
  print(board[1])
  print(board[2])

printBoard(board)
col=0
row=0
playerTurn = "X"

for counter in range(1,10):

    validMove = False
    while (validMove == False):
      col=0
      row=0
      while (col < 1 or col > 3):
        col = int(input(playerTurn + " player, select a column 1-3: "))
        if (col < 1 or col > 3):
          print("The column must be between 1 and 3.")
      while (row < 1 or row > 3):
        row = int(input(playerTurn + " player, select a row 1-3: "))
        if (row < 1 or row > 3):
          print("The row must be between 1 and 3.")
      col -= 1
      row -= 1
      if board[row][col] == '-':
        board[row][col] = playerTurn
        validMove=True;
```

```
      else:
        print("Oops, that spot was already taken. Please select another spot.")

    printBoard(board)

    if playerTurn =="X":
      playerTurn="O"
    else:
      playerTurn="X"
```

TRY IT

Directions: Try changing the lines of code to define the `printBoard()` function and the calls to it.

One of the best parts of consolidating the board output into a function is that we can change the output of the gameboard to look a bit more like the game. In the prior lesson, we looked at formatting string literals using the letter “f” before the start of the quotation mark and then with variables within the curly brackets ({}). Remember this?

```
print(f'{qty} {item} cost ${price:.2f}')
```

Another approach to formatting strings is using the string’s `.format()` method. With the **.format() method**, any character value(s) within curly brackets are replaced with the objects that are passed to the `.format()` method. The number of variables being passed in the bracket refers to the position of the object that is passed into the `.format()` method. In the example below, there are three curly brackets ({}) in the string with three variables that are passed in for each board position. Each item would be replaced in that specific order.

EXAMPLE

```
def printBoard(board):
    print("\n")
    print("\t     |     |")
    print("\t  {}  |  {}  |  {}".format(board[0][0], board[0][1], board[0][2]))
    print('\t_____|_____|_____')

    print("\t     |     |")
    print("\t  {}  |  {}  |  {}".format(board[1][0], board[1][1], board[1][2]))
    print('\t_____|_____|_____')

    print("\t     |     |")
    print("\t  {}  |  {}  |  {}".format(board[2][0], board[2][1], board[2][2]))
    print("\t     |     |")
    print("\n")
```

Now that is a much cleaner presentation that actually looks like the gameboard.

```
          |     |
       -  |  -  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
          |     |
```

`X player, select a column 1-3:`

The only lines that may be slightly new are the lines with the `.format()` method.

EXAMPLE

```
print("\t  {}  |  {}  |  {}".format(board[0][0], board[0][1], board[0][2]))
```

The `.format()` method simply replaces the {} with the variables in the order that they appear within the formatting function call. The \t is an escape character that adds in a tab. Each of the values are separated by commas. We could have also written them using a specific order by using indexes as well, like this:

```
print("\t  {0}  |  {1}  |  {2}".format(board[0][0], board[0][1], board[0][2]))
```

TRY IT

Directions: Add the improved board design into the `printboard()` function and try the updated design.

```
board = [ ["-", "-", "-"],
          ["-", "-", "-"],
          ["-", "-", "-"] ]
def printBoard(board):
   print("\n")
   print("\t     |     |")
   print("\t  {}  |  {}  |  {}".format(board[0][0], board[0][1], board[0][2]))
   print('\t_____|_____|_____')
   print("\t     |     |")
   print("\t  {}  |  {}  |  {}".format(board[1][0], board[1][1], board[1][2]))
   print('\t_____|_____|_____')
   print("\t     |     |")
   print("\t  {}  |  {}  |  {}".format(board[2][0], board[2][1], board[2][2]))
   print("\t     |     |")
   print("\n")
```

```
printBoard(board)
col=0
row=0
playerTurn = "X"
for counter in range(1,10):
   validMove = False
   while (validMove == False):
     col=0
     row=0
     while (col < 1 or col > 3):
       col = int(input(playerTurn + " player, select a column 1-3: "))
       if (col < 1 or col > 3):
         print("The column must be between 1 and 3.")
     while (row < 1 or row > 3):
       row = int(input(playerTurn + " player, select a row 1-3: "))
       if (row < 1 or row > 3):
         print("The row must be between 1 and 3.")
     col -= 1
     row -= 1
     if board[row][col] == '-':
       board[row][col] = playerTurn
       validMove=True;
     else:
       print("Oops, that spot was already taken. Please select another spot.")
   printBoard(board)
   if playerTurn =="X":
     playerTurn="O"
   else:
     playerTurn="X"
```

TERM TO KNOW

.format() method

With the `.format()` method, any character value(s) within curly brackets are replaced with the objects that are passed to the `.format()` method.

2. Checking for Winners

The next function we'll want to create is a check for the winner. There's no point in playing until we have all of the squares filled out if someone has already won.

EXAMPLE

```
def checkWinner(currPlayer, board):
  #Checking for the winner in the row
  for row in range(0,3):
    if board[row][0]==board[row][1]==board[row][2] and board[row][0] != '-':
      if board[row][0]==currPlayer:
        print("{} is winner".format(currPlayer))
        return True
  #used to check the winner in column
  for col in range(0,3):
    if board[0][col]==board[1][col]==board[2][col] and board[0][col] != '-':
      if board[0][col]==currPlayer:
        print("{} is winner".format(currPlayer))
        return True
  #used to check winner in one diagonal
  if board[0][0]==board[1][1]==board[2][2] and board[0][0] !='-':
    if board[0][0]==currPlayer:
      print("{} is winner".format(currPlayer))
      return True
  #used to check winner in another diagonal
  if board[0][2]==board[1][1]==board[2][0] and board[0][2]!='-':
    if board[0][2]==currPlayer:
      print("{} is winner".format(currPlayer))
      return True
  return False
```

In our new function called `checkWinner()`, we'll take in the `currPlayer`, whether it's an X or an O. We only need to check on the player that has made the last move. We also need to pass in the board to know its current status. Tic-Tac-Toe can only be won by either having all X's or O's in a row or a column or along each diagonal. To check each row, we will loop through each of the rows and check if they are all equal, and if they are not set to the '-' character, which is what we initialized the board with. If they are all the same (and not '-'), we will see if it matches the currPlayer, and if so, print that the currPlayer is the winner and `return True`.

EXAMPLE

```
  #Checking for the winner in the row
  for row in range(0,3):
    if board[row][0]==board[row][1]==board[row][2] and board[row][0] != '-':
      if board[row][0]==currPlayer:
        print("{} is winner".format(currPlayer))
        return True
```

Since there are three rows, we'll create a loop that loops 3 times, starting at 0 and ending prior to the index of 3:

`for row in range(0,3):`

The next line is a bit trickier, but we are essentially checking if the items in index 0, 1, and 2 are equal to each other as well as if the first item in the row is not equal to '-'. Remember that the '-' represents an empty spot.

`if board[row][0]==board[row][1]==board[row][2] and board[row][0] != '-':`

If the results are true, we'll check if that first spot is equal to the current `currPlayer`. If the first spot is equal to the current player, it means that all of them are also equal to the current player.

`if board[row][0]==currPlayer:`

If this is true, then we will output the winner to the screen and `return True`.

We will repeat this for the column. Notice that when we do that, the code looks very similar to when we check the row.

EXAMPLE

```
#used to check the winner in column
for col in range(0,3):
  if board[0][col]==board[1][col]==board[2][col] and board[0][col] != '-':
    if board[0][col]:
      print("{} is winner".format(currPlayer))
      return True
```

Afterward, we are checking for each diagonal separately.

EXAMPLE

```
#used to check winner in one diagonal
if board[0][0]==board[1][1]==board[2][2] and board[0][0] !='-':
  if board[0][0]==currPlayer:
    print("{} is winner".format(currPlayer))
    return True
#used to check winner in another diagonal
if board[0][2]==board[1][1]==board[2][0] and board[0][2]!='-':
  if board[0][2]==currPlayer:
    print("{} is winner".format(currPlayer))
    return True
```

Using that, once we print the board, we can `break` out of the loop since the individual has won.

EXAMPLE

```
  printBoard(board)
```

```
    if (checkWinner(playerTurn,board)):
      break
```

Let's see what our final code looks like in full.

EXAMPLE

```
board = [ ["-", "-", "-"],
          ["-", "-", "-"],
          ["-", "-", "-"] ]

#printing the board
def printBoard(board):
    print("\n")
    print("\t     |     |")
    print("\t  {}  |  {}  |  {}".format(board[0][0], board[0][1], board[0][2]))
    print('\t_____|_____|_____')

    print("\t     |     |")
    print("\t  {}  |  {}  |  {}".format(board[1][0], board[1][1], board[1][2]))
    print('\t_____|_____|_____')

    print("\t     |     |")
    print("\t  {}  |  {}  |  {}".format(board[2][0], board[2][1], board[2][2]))
    print("\t     |     |")
    print("\n")

def checkWinner(currPlayer, board):
  #Checking for the winner in the row
  for row in range(0,3):
    if board[row][0]==board[row][1]==board[row][2] and board[row][0] != '-':
      if board[row][0]==currPlayer:
        print("{} is winner".format(currPlayer))
        return True
  #used to check the winner in column
  for col in range(0,3):
    if board[0][col]==board[1][col]==board[2][col] and board[0][col] != '-':
      if board[0][col]==currPlayer:
        print("{} is winner".format(currPlayer))
        return True
  #used to check winner in one diagonal
  if board[0][0]==board[1][1]==board[2][2] and board[0][0] !='-':
    if board[0][0]==currPlayer:
      print("{} is winner".format(currPlayer))
```

```
      return True
  #used to check winner in another diagonal
  if board[0][2]==board[1][1]==board[2][0] and board[0][2]!='-':
    if board[0][2]==currPlayer:
      print("{} is winner".format(currPlayer))
      return True
  return False

printBoard(board)
col=0
row=0
playerTurn = "X"

for counter in range(1,10):
    validMove = False
    while (validMove == False):
      col=0
      row=0
      while (col < 1 or col > 3):
        col = int(input(playerTurn + " player, select a column 1-3: "))
        if (col < 1 or col > 3):
          print("The column must be between 1 and 3.")
      while (row < 1 or row > 3):
        row = int(input(playerTurn + " player, select a row 1-3: "))
        if (row < 1 or row > 3):
          print("The row must be between 1 and 3.")
      col -= 1
      row -= 1
      if board[row][col] == '-':
        board[row][col] = playerTurn
        validMove=True;
      else:
        print("Oops, that spot was already taken. Please select another spot.")
        validMove=False
        row=0
        col=0

    printBoard(board)
    if (checkWinner(playerTurn,board)):
      break

    if playerTurn =="X":
```

```
      playerTurn="O"
    else:
      playerTurn="X"
```

Here we are testing out a complete game:

```
          |     |
       -  |  -  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
          |     |

X player, select a column 1-3:  1
X player, select a row 1-3:  1

          |     |
       X  |  -  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
          |     |

O player, select a column 1-3:  1
O player, select a row 1-3:  2

          |     |
       X  |  -  |  -
     _____|_____|_____
          |     |
       O  |  -  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
          |     |
```

```
X player, select a column 1-3:  2
X player, select a row 1-3:  2

          |     |
       X  |  -  |  -
     _____|_____|_____
          |     |
       O  |  X  |  -
     _____|_____|_____
          |     |
       -  |  -  |  -
          |     |

O player, select a column 1-3:  1
O player, select a row 1-3:  3

          |     |
       X  |  -  |  -
     _____|_____|_____
          |     |
       O  |  X  |  -
     _____|_____|_____
          |     |
       O  |  -  |  -
          |     |

X player, select a column 1-3:  3
X player, select a row 1-3:  3

          |     |
       X  |  -  |  -
     _____|_____|_____
          |     |
       O  |  X  |  -
     _____|_____|_____
          |     |
       O  |  -  |  X
          |     |
```

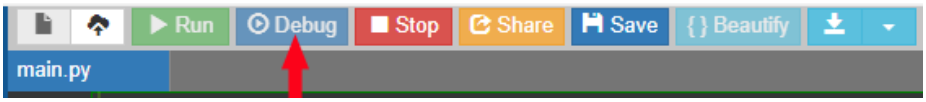
```
X is winner
```

Debugging this game

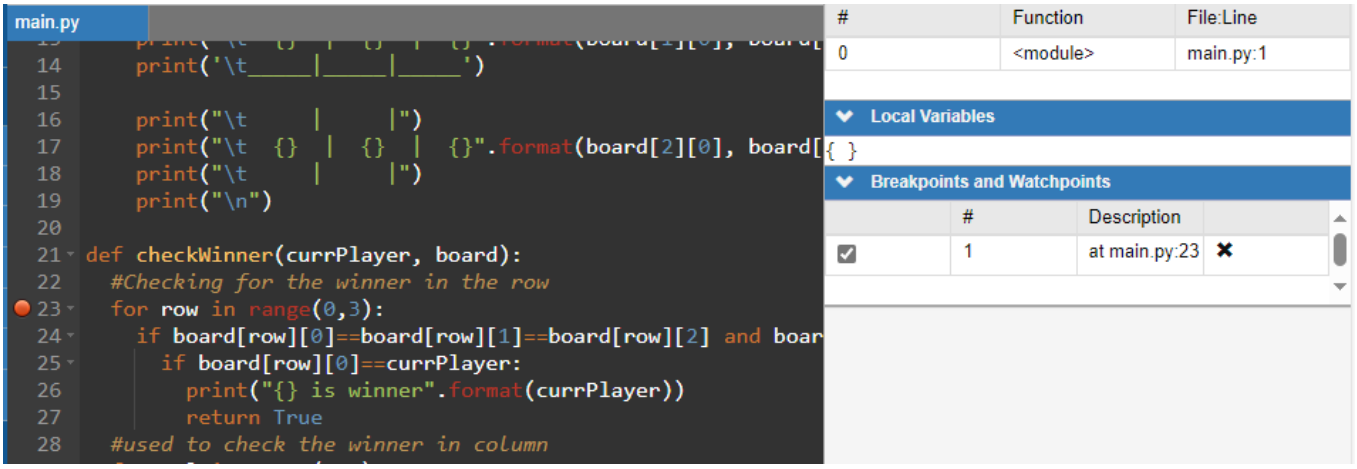
This is also a good time to come back to the GDB debug functionality to test out what happens when the checking of the winner takes place in the `checkWinner()` function. Remember that we used the debug functions within GDB back in Unit 1 and that we will revisit the tool again in future lessons.

TRY IT

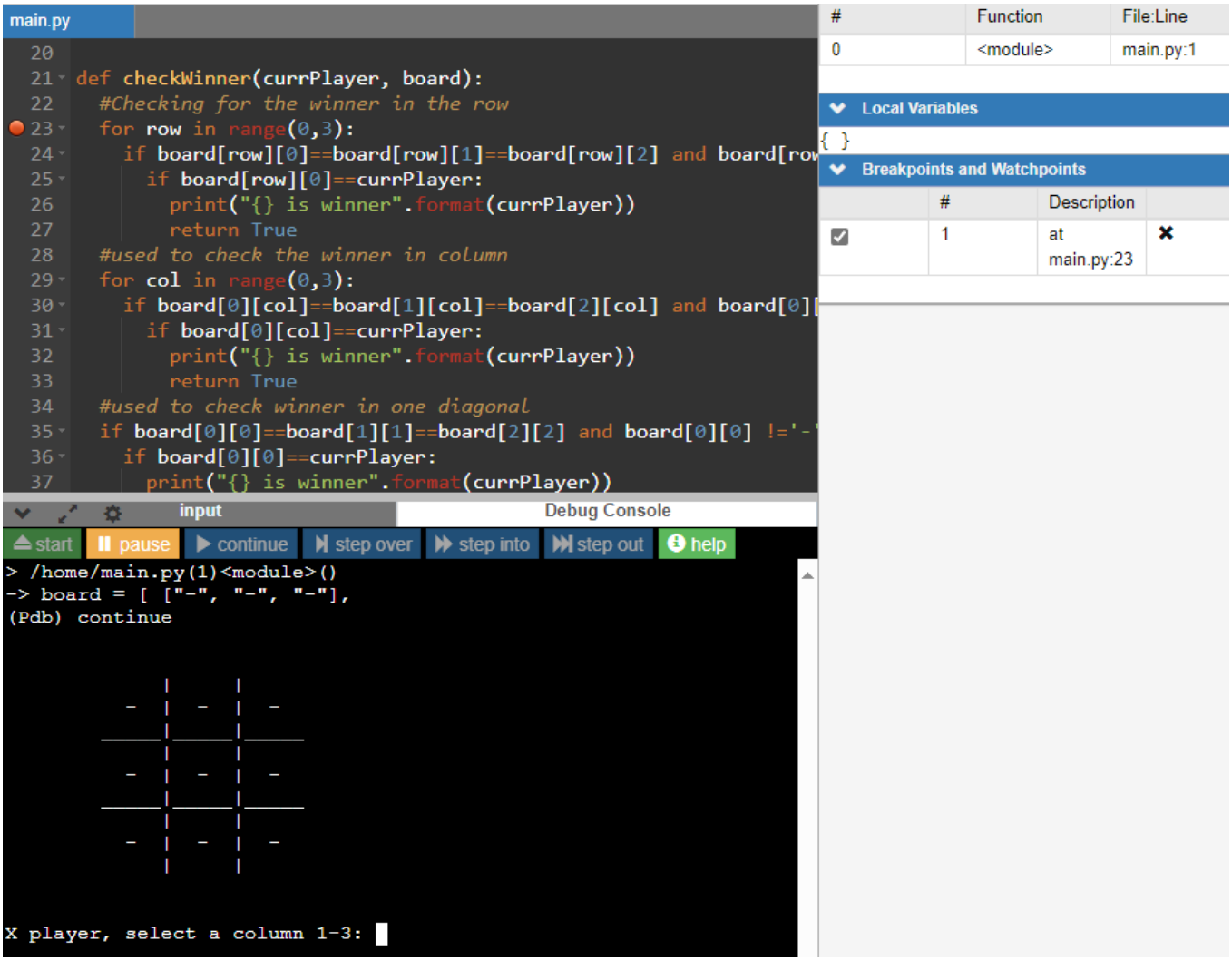
Directions: Make sure you have added the code into a GDB project. Then follow the following actions to step through the program. To start, we'll access the debug function by clicking on "Debug" in the top menu.



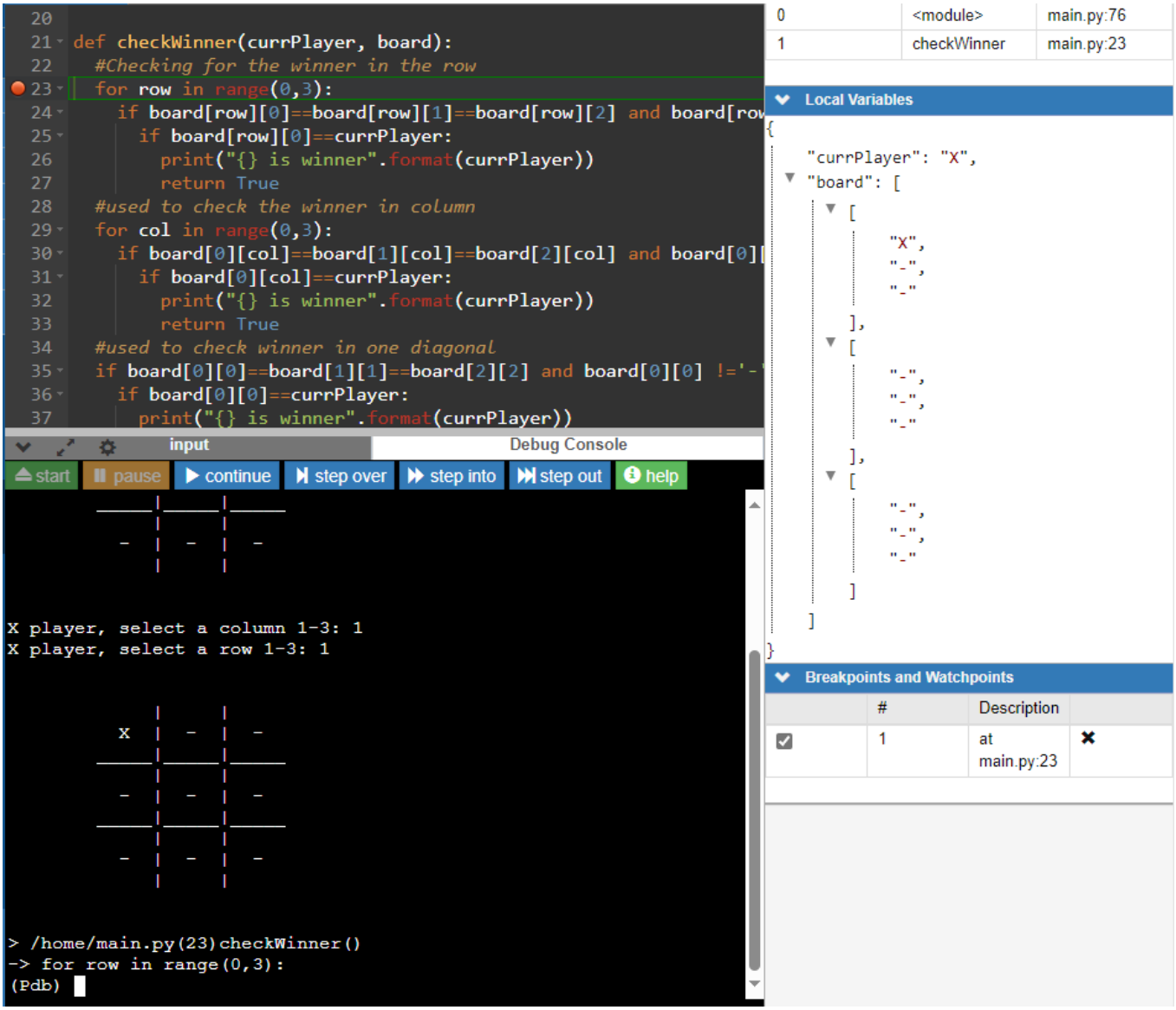
On the code editor pane, go to the first line of code in the `checkWinner()` function. We'll add a breakpoint by clicking to the left of the line number. This line number may be different depending on any changes you may have made, but it should be set to the line with the first `for` loop where we're checking for the winner in the row.



We'll go ahead and start the debug process by pressing the "Continue" button in the Debug Console.



As you can see, the program paused on the user input for the column and row for the “X” player. Once that information is added to the Debug Console, the program will execute until the `checkWinner()` function is called, and stop on the `for` loop (our identified breakpoint).

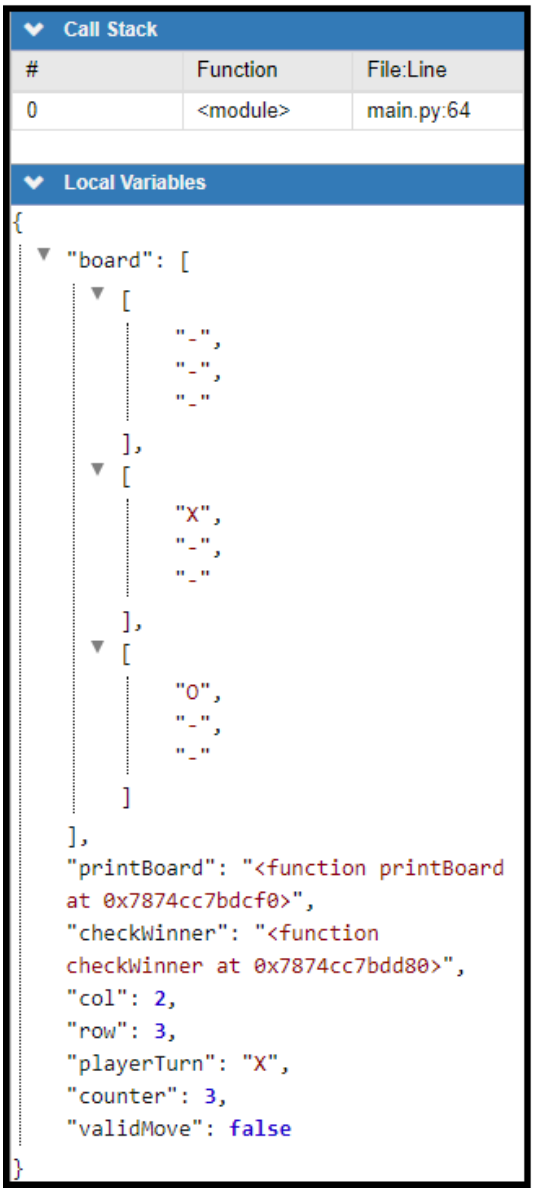


Notice that we have some added information in the status pane (right-side area) under Local Variables. This will show you what the values of variables are as you move through the program.

Let's step through the code one line at a time to see the results. Given that there's only one item placed, we should expect to see that no winner will be announced. Clicking on the "Step Over" button will allow you to step through one line at a time.

Note: Watch as each line of code is called. It will progress for many steps before it returns to the breakpoint.

As you are stepping through, you'll be able to track the values of the variables like the row being set to 0, then on the next iteration (next player's input) of the loop, it being set to another number (like in the image below) based on what the player selected.



You can continue to step through the code like this, and once you're ready to get to the next iteration, you can trim down the amount of Debug Console outputs by clicking on the "Continue" button. Remember the "Continue" button resumes program execution until next breakpoint is reached. However, it still needs to check all ranges so it will take about four steps to completely return to the breakpoint to ask for the next players selection.

TRY IT

Directions: This is the culmination of three challenges on revising this program. Go ahead and play a few games. Play with a family member or friend. Try using the debug function again by adding more breakpoints and watching the variables change by stepping through the program.

Now that we're all done with the Tic-Tac-Toe game, try to think about how else you can improve the functionality of the game. If you wanted to have the players play multiple games, or if you wanted to keep track of who won, how would you do that?

To see the final version of this program visit **Sophia's Python code page**

SUMMARY

In this lesson, we learned about adding more complex functions to the Tic-Tac-Toe game. We added a function to **improve the board** by enhancing its look to resemble an actual Tic-Tac-Toe board. We also added a function to **check on the winner** of the game. As part of the `checkWinner()` function, we checked each row, each column, and both diagonals for a possible winner and if found, we broke out of the loop to declare the winner. Finally, we revisited the debug functionality in the IDE by adding a breakpoint and stepped through the program's execution while watching the variables change in the status pane.

Best of luck in your learning!

TERMS TO KNOW

.format() method
With the .format() method, any character value(s) within curly brackets are replaced with the objects that are passed to the .format() method.