Girls' Programming Network

Tic-Tac-Toe

Tutors Only

Create a 2 player Tic Tac Toe game to play with your friends!

Part 0: Setting up

Task 0.1: Making a python file [and opening IDLE]

- Make sure the student selects **IDLE Python 3.**
  Other versions of Python may not open, or may cause confusion due to Python 2/3 differences.
- Make sure the student ends the filename with **.py**. Otherwise, the helpful syntax formatting colours may not appear in the IDLE editor.

Task 0.2: You've got a blank space, so write your name!

Encourage the student to write her name at the top of the file, in a comment:

```
1    # Firstname Lastname
2
3
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 1:

☐ You should have a file called tic_tac_toe.py

☐ Your file has your name at the top in a comment

☐ Run your file with F5 key and it does nothing!!

★ BONUS 0.3: Customised Welcome ★

- If the student's program can't find the *name* variable, check that they've spelled the variable in the same way, and that the capitalization is the same.

Part 1: Welcome to Tic-Tac-Toe!

Task 1.1: Storing the board

The student's code should look like this:

```
# Firstname Lastname
board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
```

Hint

Make sure the student has called the list `board`, as opposed to `my_list`.

Task 1.2: Printing the board

At this stage, the code should look like this:

```
# Firstname Lastname
board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
print("------------")
print("|", board[0], "|", board[1], "|", board[2], "|")
print("------------")
print("|", board[3], "|", board[4], "|", board[5], "|")
print("------------")
print("|", board[6], "|", board[7], "|", board[8], "|")
print("------------")
```

Hint: Concatenation

Students may need to be reminded that you can put multiple things in a print statement, separated by commas, and that they will have spaces inserted between them.

Task 1.3: Print Test!

After testing their board with different symbols in it, make sure students reset it to just having spaces.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 2:

☐ Use a list to create the board

☐ Print your empty playing board

Part 2: Enter The First Move

Task 2.1: What symbol are you?

The code should look like this:

```python
# Firstname Lastname
board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
print("-------------")
print("|", board[0], "|", board[1], "|", board[2], "|")
print("-------------")
print("|", board[3], "|", board[4], "|", board[5], "|")
print("-------------")
print("|", board[6], "|", board[7], "|", board[8], "|")
print("-------------")

symbol = "O"
```

Task 2.2: Which spot do you want to choose?

The student's code should be as above, plus this line:

```python
square = input("Which square do you want your symbol to go in?")
```

Hint

Leave a space after the question mark, so that the prompt isn't immediately followed by the user's input (as seen here):

```
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Which square do you want your symbol to go in?oh no there's no space!
```

Needs a space!

Task 2.3: Find the square on the board

The code should be the same as above, but the square question becomes:

```python
square = input("Which square do you want your symbol to go in?")
square_index = int(square)
```

Hint

Some students may combine the two lines into one, which is fine:

```
12    square_index = int(input("Which square do you want your symbol
          to go in?"))
```

However, if they name the variable **square** (instead of **square_index**), they could get confused when later instructions refer to **square_index**.

Task 2.4: Update list with player's symbol

The student's full code should now be:

```
1    # Firstname Lastname
2    board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
3    print("-------------")
4    print("|", board[0], "|", board[1], "|", board[2], "|")
5    print("-------------")
6    print("|", board[3], "|", board[4], "|", board[5], "|")
7    print("-------------")
8    print("|", board[6], "|", board[7], "|", board[8], "|")
9    print("-------------")
10
11   symbol = "O"
12   square = input("Which square do you want your symbol to go in?")
13   square_index = int(square)
14   board[square_index] = symbol
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 3:

☐ The player's chosen symbol is stored in a variable

☐ The spot the player wants to move is stored in a variable

☐ Update the list with player's symbol

★ BONUS 2.5: Welcome the players ★

If the student does this bonus, the code should look something like this:

```
player_O = input("Who is playing naughts? ")
player_X = input("And who is playing crosses? ")
print("Welcome", player_O, ", your symbol is O!")
print("And welcome", player_X, ", your symbol is X!")
```

Part 3: Creating a print function

Functions

We updated the **board** list, but to actually show our updated board, we need to **print** the board again. We already have code that **print**s, so to avoid repeating that code, let's create a function!

Task 3.1: Define your function!

The top of the student's code should now look like this:

```
# Firstname Lastname

def print_board(board):

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

Task 3.2: Fill in your function body

The code should look like this:

```
 1   # Firstname Lastname
 2
 3
 4   def print_board(board):
 5       print("-------------")
 6       print("|", board[0], "|", board[1], "|", board[2], "|")
 7       print("-------------")
 8       print("|", board[3], "|", board[4], "|", board[5], "|")
 9       print("-------------")
10       print("|", board[6], "|", board[7], "|", board[8], "|")
11       print("-------------")
```

Hint: What is the function body?

- Make sure that the line assigning the **board** variable **is not inside the function.**
  The board should be initialized separately, outside of the function.
  I'm referring to this line:

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

- Students may have **indenting issues** here! If they've indented their code with the spacebar instead of the tab key, look carefully - one of their indented lines may have 3 spaces instead of 4.

Task 3.3: Let's call our function!

After this step, the code should now look like this:

```
 1   # Firstname Lastname
 2
 3   def print_board(board):
 4       print("-------------")
 5       print("|", board[0], "|", board[1], "|", board[2], "|")
 6       print("-------------")
 7       print("|", board[3], "|", board[4], "|", board[5], "|")
 8       print("-------------")
 9       print("|", board[6], "|", board[7], "|", board[8], "|")
10       print("-------------")
11
12   board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
13   print_board(board)          ⟵——— newly added
14
15   # BONUS 2.5 may be here, if the student did it
16
17   symbol = "O"
18   square = input("Which square do you want your symbol to go in?")
19   square_index = int(square)
20   board[square_index] = symbol
21
```

Task 3.4: Reveal the updated board!

The code should now be as above, plus a final line that calls the function again:

```
print_board(board)
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 4:

☐ Define a function called `print_board`

☐ Print your empty playing board at the beginning of your game

☐ Print your updated playing board after your first move

Part 4: Taking Turns

Task 4.1: Your turn!

The student should add a print statement after they set the symbol:

```
17    symbol = "O"
18    print("The current player is", symbol, "!")
19    square = input("Which square do you want your symbol to go in?")
20    square_index = int(square)
21    board[square_index] = symbol
22
23    print_board(board)
```

new line

Task 4.2: *You* get a turn! And *you* get a turn! And *you* get a turn!

The student's code should be as above, plus this **if** statement at the end:

```
24
25 ▼  if symbol == "O":
26         symbol = "X"
```

Task 4.3: Switch back!

The **if** statement now gets a second part to switch back to naughts:

```
25    if symbol == "O":
26        symbol = "X"
27 ▼  else:
28        symbol = "O"
```

WARNING! If the student writes **if** twice, both conditions will be true! The symbol will be changed to crosses, then immediately back to naughts:

```
25    if symbol == "O":
26        symbol = "X"
27    if symbol == "X":
28        symbol = "O"
```
Wrong! Don't do this!

So be sure to use either **else** or **elif**, so that the symbol is only changed once.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 5:

☐ Start playing as Noughts

☐ Tell the players whose turn it is

☐ Switch players at the bottom of your code

Part 5: Wait a while to win

Task 5.1: Game Over?

After initializing the board, the student's code should initialize a new variable:

```
10        print("-------------")
11
12     board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
13     game_over = False    ← ─ line added
14     print_board(board)
15
```

Task 5.2: Did I win yet?

After adding the while loop, the code should look like this (starting from the board initialization):

```
11
12     board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
13     game_over = False
14     print_board(board)
15
16     # BONUS 2.5 may be here, if the student did it
17
18     symbol = "O"
19     while not game_over:
20         print("The current player is", symbol, "!")
21         square = input("Which square do you want your symbol to go in?")
22         square_index = int(square)
23         board[square_index] = symbol
24
25         print_board(board)
26
27         if symbol == "O":
28             symbol = "X"
29         if symbol == "X":
30             symbol = "O"
31
32         else:
33             symbol = "O"
34
```

All this code is indented now

Hint

Make sure the while loop condition is **not** `game_over` - don't miss the **not** !

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 6:
☐ You have set your `game_over` variable
☐ You have built your while loop
☐ You have run your code to make sure the indents are correct and get no errors

Part 6: Winner winner, tic tac dinner!

Task 6.1: Where are the winners

The answers for this section are:

Rows	0, 1, 2	3, 4, 5	6, 7, 8
Columns	0, 3, 6	1, 4, 7	2, 5, 8
Diagonals	0, 4, 8	2, 4, 6	--

Task 6.2: Functions again!

The top of the student's code should get a new function definition:

```
1    # Firstname Lastname
2
3    def check_winner(board):
4
5
6 ▼  def print_board(board):
7        print("-------------")
8        print("|", board[0], "|", board[
```

↰ new function started

Task 6.3: One function, two options!

The student can fill in check_winner(board) in two ways, (1) Blue or (2) Orange. Depending on their choice, their function should look something like this:

Option 1: If statements

```python
# Blue version
def check_winner(board):
    if board[0] == board[1] == board[2] != " ":
        return True
    elif board[3] == board[4] == board[5] != " ":
        return True
    elif board[6] == board[7] == board[8] != " ":
        return True
    elif board[0] == board[3] == board[6] != " ":
        return True
    elif board[1] == board[4] == board[7] != " ":
        return True
    elif board[2] == board[5] == board[8] != " ":
        return True
    elif board[0] == board[4] == board[8] != " ":
        return True
    elif board[2] == board[4] == board[6] != " ":
        return True
    else:
        return False
```

Note that the triples are exactly the same as the winning triples identified in the last exercise. **Don't forget to check that the triples aren't spaces!**

Option 2: For loop and lists

```python
# Orange version
def check_winner(board):
    winning_combos = [
        # Rows
        (0,1,2),
        (3,4,5),
        (6,7,8),
        # Columns
        (0,3,6),
        (1,4,7),
        (2,5,8),
        # Diagonals
        (0,4,8),
        (2,4,6),
    ]
    for combo in winning_combos:
        combo_part_0 = combo[0]
        combo_part_1 = combo[1]
        combo_part_2 = combo[2]
        symbol_0 = board[combo_part_0]
        symbol_1 = board[combo_part_1]
        symbol_2 = board[combo_part_2]
        if symbol_0 == symbol_1 == symbol_2 != " ":
            return True
    return False
```

Again, notice how the triples are the same as those identified in the last exercise.

☑ CHECKPOINT ☑

OPTION 1: If you can tick all of these off you can go to Part 7:

☐ You have created the `check_winner` function

☐ You return the result from the `check_winner` function

☐ Your `check_winner` function checks every possible way to win

☑ CHECKPOINT ☑

OPTION 2: If you can tick all of these off you can go to Part 7:

☐ You have created `winning_combos` with eight tuples in it

☐ You have created the `check_winner` function

☐ You return the result from the `check_winner` function

☐ Your `check_winner` function checks every possible way to win

Part 7: Declare the winner

Task 7.1: Check whether the game has been won

The main game loop should now look like this:

```python
while not game_over:
    print("The current player is", symbol, "!")
    square = input("Which square do you want your symbol to go in?")
    square_index = int(square)
    board[square_index] = symbol

    print_board(board)
    game_over = check_winner(board)

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

Task 7.2: Declare who won

The final code for the base game is as follows:

```python
1   # Firstname Lastname
2
3   # Blue version of check_winner()
4   def check_winner(board):
5       if board[0] == board[1] == board[2] != " ":
6           return True
7       elif board[3] == board[4] == board[5] != " ":
8           return True
9       elif board[6] == board[7] == board[8] != " ":
10          return True
11      elif board[0] == board[3] == board[6] != " ":
12          return True
13      elif board[1] == board[4] == board[7] != " ":
14          return True
15      elif board[2] == board[5] == board[8] != " ":
16          return True
17      elif board[0] == board[4] == board[8] != " ":
18          return True
19      elif board[2] == board[4] == board[6] != " ":
20          return True
21      else:
22          return False
23
```

```python
24   # Orange version of check_winner()
25 ▼ def check_winner(board):
26 ▼     winning_combos = [
27           # Rows
28           (0,1,2),
29           (3,4,5),
30           (6,7,8),
31           # Columns
32           (0,3,6),
33           (1,4,7),
34           (2,5,8),
35           # Diagonals
36           (0,4,8),
37           (2,4,6),
38       ]
39 ▼     for combo in winning_combos:
40           combo_part_0 = combo[0]
41           combo_part_1 = combo[1]
42           combo_part_2 = combo[2]
43           symbol_0 = board[combo_part_0]
44           symbol_1 = board[combo_part_1]
45           symbol_2 = board[combo_part_2]
46           if symbol_0 == symbol_1 == symbol_2 != " ":
47               return True
48       return False
49

50 ▼ def print_board(board):
51       print("-------------")
52       print("|", board[0], "|", board[1], "|", board[2], "|")
53       print("-------------")
54       print("|", board[3], "|", board[4], "|", board[5], "|")
55       print("-------------")
56       print("|", board[6], "|", board[7], "|", board[8], "|")
57       print("-------------")
58
59   board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
60   game_over = False
61   print_board(board)
62
63   # BONUS 2.5 may be here, if the student did it
64
65   symbol = "O"

66   while not game_over:
67       print("The current player is", symbol, "!")
68       square = input("Which square do you want your symbol to go in?")
69       square_index = int(square)
70       board[square_index] = symbol
71
72       print_board(board)
73       game_over = check_winner(board)
74       if game_over:
75           print(symbol, "won! Congratulations!")
76
77       if symbol == "O":
78           symbol = "X"
79       else:
80           symbol = "O"
```

Girls' Programming Network

Tic-Tac-Toe

Extensions

TUTORS ONLY

Make your 2 player Tic Tac Toe game even better!

Extension 8: You can't go there!

At the moment the game lets you cheat by playing in a spot someone else has already taken! And the game breaks if you enter a number bigger than 8!

Let's fix it so you can only play in spots that actually exist! And not ones that are taken.

Task 8.1: What spaces are free?

The answers for this exercise are:

X	O	X

free_squares =
[3,4,5,6,7,8]

O		
	X	
		O

free_squares =
[1,2,3,5,6,7]

	X	X
	O	O

free_squares =
[2,5,6,7,8]

Task 8.2: What spaces are free?

The student should initialize the list just before the main loop:

```
free_squares = [0,1,2,3,4,5,6,7,8]
while not game_over:
    print("The current player is", symbol, "!")
    square = input("Which square do you want your symbol
    square_index = int(square)
```

Task 8.3: Keeping track of spaces

Inside the main game loop, the student needs to add two lines after they print the board:

```python
while not game_over:
    print("The current player is", symbol, "!")
    square = input("Which square do you want your symbol to go in?")
    square_index = int(square)
    board[square_index] = symbol

    print_board(board)
    free_squares.remove(square_index)   ⟵
    print(free_squares)                 ⟵
    game_over = check_winner(board)
    if game_over:
        print(symbol, "won! Congratulations!")
```

Task 8.4: Is that even allowed?

At the end of this task, the student should have added these two lines:

```python
    square_index = int(square)
    if not square_index in free_squares:
        print("Hey! That square is not allowed!")
    board[square_index] = symbol
```

HINT: These two versions of the line do exactly the same thing:

```python
if square_index not in free_squares:
```

```python
if not square_index in free_squares:
```

Task 8.5: Try that again

The final code for Extension 8 has the main game loop as follows, with the new parts highlighted:

```python
symbol = "O"
free_squares = [0,1,2,3,4,5,6,7,8]
while not game_over:
    print("The current player is", symbol, "!")
    square = input("Which square do you want your symbol to go in?")
    square_index = int(square)
    if not square_index in free_squares:
        print("Hey! That square is not allowed!")
        continue
    board[square_index] = symbol

    print_board(board)
    free_squares.remove(square_index)
    print(free_squares)
    game_over = check_winner(board)
    if game_over:
        print(symbol, "won! Congratulations!")

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

☑ CHECKPOINT ☑

If you can tick all of these off you've finished Extension 8:

☐ Your game doesn't let you play in a square someone already filled

☐ Your game tells the player if they chose a bad square and starts their turn again

☐ Your game doesn't break when you choose squares like 99 or -5

Extension 9: It's a tie!

When we hit 9 moves, the board is filled and there's no winner. We have a tie!

Task 9.1: Count the turns

Tutor note: The example code given assumes the student is *only* doing Extension 9, without doing Extension 8 first.

The student needs to add two lines for this task:

```python
symbol = "O"
counter = 0
while not game_over:
    print("The current player is", symbol, "!")
    square = input("Which square do you want your symbol to go in?")
    square_index = int(square)
    board[square_index] = symbol

    print_board(board)
    counter = counter + 1
    game_over = check_winner(board)
    if game_over:
        print(symbol, "won! Congratulations!")

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

Task 9.2: Check for a tie

After this task, the code should look like this (new lines highlighted):

```python
symbol = "O"
counter = 0
while not game_over:
    print("The current player is", symbol, "!")
    square = input("Which square do you want your symbol to go in?")
    square_index = int(square)
    board[square_index] = symbol

    print_board(board)
    counter = counter + 1
    game_over = check_winner(board)
    if game_over:
        print(symbol, "won! Congratulations!")
    elif counter == 9:
        print("It's a tie!")
        break

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

☑ CHECKPOINT ☑

If you can tick all of these off you've finished Extension 9:

☐ The game ends if the board is all filled up

☐ The game prints that it's a tie if no one has won at the end of the game

Extension 10: Coin Toss

At the moment the same symbol always starts first. Let's make it randomly chooses who goes first!

Task 10.1: This is random!

Tutor note: The code snippets are assuming that the student is *only* doing Extension 10, and hasn't done Extension 8 or 9.

This task is super easy, just add the import at the top of the file:

```
1   # Firstname Lastname
2   import random
3
```

Task 10.2: Who will go first?

The student can solve this task with **random.choice()**:

```
symbol = random.choice(["O", "X"])
```

Task 10.3: Tell us who's next!

Depending on their existing code, **the student may not need to change their program at all** in order to announce the next player:

```
symbol = random.choice(["O", "X"])
while not game_over:
    print("The current player is", symbol, "!")
    square = input("Which square do you want your symbol to go in?")
    square_index = int(square)
    board[square_index] = symbol
```

They probably have this line already (handwritten annotation pointing to the print line)

Otherwise, a **print** statement will do the job.

☑ CHECKPOINT ☑

If you can tick all of these off you've finished Extension 10:

☐ Your game randomly chooses which symbol will start

☐ Your game announces the winner of the coin toss

Extension 11: A game that knows your name!

It would be better if the game actually referred to you by name, not just your symbol!

Task 11.1: Prepare yourself!

Tutor note: These code snippets assume that the student already completed Extension 10.

Also, make sure the student has done **Bonus 2.5** where the program asks for the players names.

Task 11.2: Who's there

After the function definitions, but **before** the main game loop, the student's code should now look something like this:

```python
60    board = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
61    game_over = False
62    print_board(board)
63
64    # BONUS 2.5 starts
65    player_O = input("Who is playing naughts? ")
66    player_X = input("Who is playing crosses? ")
67    print("Welcome", player_O, ", your symbol is O!")
68    print("And welcome", player_X, ", your symbol is X!")
69    # BONUS 2.5 ends
70
71    symbol = random.choice(["O", "X"])
72    if symbol == "O":
73        current_player = player_O
74    else:
75        current_player = player_X
76
```

Task 11.3: It's your turn!

Before this task, the code would have been:

```python
while not game_over:
    print("The current player is", symbol, "!")
```

After this task, the code becomes:

Line changed

```python
while not game_over:
    print("The current player is", current_player, ", who is playing the
        symbol", symbol, "!")
```

Task 11.4: Who's next?

Inside the main game loop, the student can take advantage of the existing if statements, changing the **current_player** at the same time as the **symbol**:

```python
if symbol == "O":
    symbol = "X"
    current_player = player_X
else:
    symbol = "O"
    current_player = player_O
```

Task 11.5: Who's won?

This one is simple, the student just needs to change the **symbol** variable to **current_player**:

```python
77    while not game_over:
78        print("The current player is", current_player, ", who is playing the
```

```python
84        game_over = check_winner(board)
85        if game_over:
86            print(current_player, "won! Congratulations!")
```

change the variable

☑ CHECKPOINT ☑

If you can tick all of these off you've finished Extension 11:

☐ The game prints out the name of the player who owns the symbol each turn

☐ The game keeps track of which players turn it is.

Extension 12: Random computer player

Right now we need a friend to play, but what if we want to play when no one else is around? Let's make very basic computer player. It will randomly choose a place to put its symbols!

In this game if one of the names entered is computer, then we will choose a random square for the computer to fill. (I hope none of your friends names are computer!)

Task 12.1: Prepare yourself!

Make sure the student has done the other extensions:
1. **Extension 8: You Can't Go There**
2. **Extension 10: Coin Toss**
3. **Extension 11: A Game that Knows Your Name!**

The code snippets below assume that the student has completed those extensions.

Task 12.2: My name is computer

The student needs to add an `if` statement which checks if it's the computer's turn. If so, the computer chooses a free square randomly:

```
77  free_squares = [0,1,2,3,4,5,6,7,8]
78  while not game_over:
79      print("The current player is", current_player, ", who is playing
            the symbol", symbol, "!")
80
81      if current_player == "computer":
82          square = random.choice(free_squares)       ← Added lines
83
84      square = input("Which square do you want your symbol to go in?")
85      square_index = int(square)
```

Task 12.3: I'm no robot!

Make sure that the student **only** indents the code that asks the human player for their square!
The code that places the symbol on the board should still be out of the `if/else` block:

```
while not game_over:
    print("The current player is", current_player, ", who is playing
        the symbol", symbol, "!")

    if current_player == "computer":
        square = random.choice(free_squares)
    else:          ← — newly indented block
        square = input("Which square do you want your symbol to go in?")

    square_index = int(square)
    if square_index not in free_squares:
        print("That square isn't allowed!")
        continue

    board[square_index] = symbol
    free_squares.remove(square_index)
```

This remains outside the indented block

☑ CHECKPOINT ☑

If you can tick all of these off you've finished Extension 12:

☐ If you say a computer is playing the game randomly chooses moves for the computers turn.

☐ You print out the free squares each turn and it gets smaller as the game goes on.

☐ The human player still gets to choose a move on their turn.

★ BONUS 12.4: Smarter Computer ★

BONUS 12.4 can be approached in multiple ways. The student will try to code:
- A computer player that will play in an empty spot that would complete their line
- Will block the opponent from winning if the opponent already has two in a row
- All the other times it can still play randomly

General debugging principles are useful here; if the student is stuck, encourage her to add print statements for important variables, to check that everything is working as she expects.