# Girls' Programming Network

# *Tic-Tac-Toe*

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

# Part 0: Setting up

Intro to Python

## Task 0.1: Making a python file

Open the start menu, and type 'IDLE'. Select IDLE Python 3.

1. Go to the file menu and select 'New File'. This opens a new window.
2. Go to the file menu, select 'Save As'
3. Choose Desktop and save the file as 'tic_tac_toe.py'

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

At the top of the file use a comment to write your name!
Any line starting with # is a comment.

```
# This is a comment
```

## ☑ 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 ★

**Waiting for the next lecture? Try adding this bonus feature!!**

1. Ask the user for their name.

2. Print a customised message for the user using their name. Something like this:

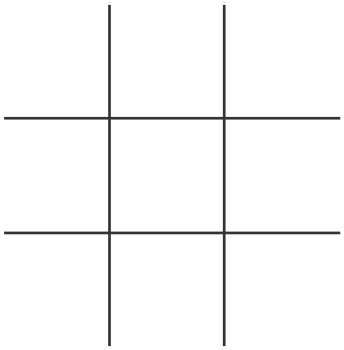
```
----------------------------------------
    Welcome to Tic-Tac-Toe, Lyndsey
----------------------------------------
```

Lists

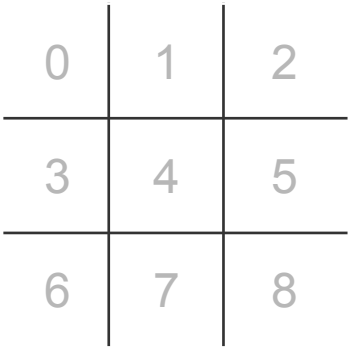
# Part 1: Welcome to Tic-Tac-Toe!

### The Tic Tac Toe board!

The board we're going to play on will look like this when we start.



But we need to make up names we'll call each square. Here's how we'll number the board **in our heads**.




**Why start from 0?**
**Computers count from 0. This will help us later!**

## Task 1.1: Storing the board

To keep track of where moves have been made on the board, we're going to use a list. To start with, represent each square on the board as a blank space.

1. Create a list called **`board`** containing 9 items. Each of the items should be a single space character like this: `" "`

## *Hint*

Here's an example of a list:

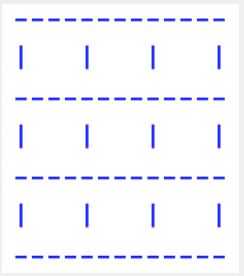
```
my_list = ["item1", " ", "item3", "etc"]
```

## Task 1.2: Printing the board

**We need to print out the data in the board!**

1. **Using the data in your list**, `print` out a grid to represent the **`board`**.
    - Don't forget to add in the symbols that are stored in your board list!
    - There should be 13 dashes on the flat lines.
    - You can get the vertical lines (called pipes) by pressing Shift-\, just under the Backspace key. Add a space between each pipe and board symbol

When the **`board`** only has `" "` in it, it should look like this:




### *Hint: Printing out a specific item in a list*

The below code prints the second item in our list of favourite foods:

```
fave_foods = ["pie", "ice cream", "banana"]
print(fave_foods[1])
```

## Task 1.3: Print Test!

See if your board printing code works when you have symbols in your list!

1. Try manually setting your starting board to have some Xs and Os in it, instead of all spaces.

2. Run your code! Do they print out in the board like you would expect?

3. Afterwards, reset your starting board to just have spaces in it.

## ☑ 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?

We can have the user enter their own symbol (number, letter, anything they want!) for the board.

1. Create a variable called **`symbol`** and set it to `"O"` initially.

## Task 2.2: Which spot do you want to choose?

Now we need to know which square to put the symbol in.

1. Ask the user for the number of the square they want their symbol to go in, and store it in a variable called **`square`**.

### *Hint*

*Remember* this is how we get the computer to ask a question and store the answer:

```
your_name = input("What is your name?")
```

## Task 2.3: Find the square on the board

Right now, the **`square`** variable stores the number the user entered, but as a `"string"`. We need to change it to an integer so we can use it to look up items in our list.

1. Convert **`square`** to an integer, and store it in a new variable called **`square_index`**.

### *Hint*

Here's how we can turn variables into integers:

```
my_num = int(my_num)
```

## Task 2.4: Update list with player's symbol

Now we need to update our list with the player's symbol so it's not a blank space.

1. Update the selected **`square_index`** in the **`board`** list so it contains the **`symbol`**.

### *Hint*

Here's how you can update a specific item in a list:

```
mylist[3] = 5
```

## ☑ 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

☐ Nothing is printed after the square is selected (we'll do that next!)

## ★ BONUS 2.5: Welcome the players ★

**Waiting for the next lecture? Try adding this bonus feature!!**

Welcome the players and announce what symbols they will be playing

1. Ask for the name of the player who will be playing as naughts. Store the name in a variable called `player_O`
2. Ask for the name of the player who will be playing as crosses. Store the name in a variable called `player_X`
3. Print out a welcome. Announce each player and what symbol they are using.

# Part 3: Creating a print function

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!

1. At the top of your code, under the comment with your name, define a function called **print_board**. It will take a single argument, called **board**.

### *Hint: Defining a function*

If we wanted to define a new function that:

- Was called **add_num**; and
- Took in **num1** and **num2** as arguments

It would look like this:

```
def add_num(num1, num2):
```

## Task 3.2: Fill in your function body

1. Find the **print** statements that you added in Task 1.3 and copy these statements to your function body.

### *Hint: What is the function body?*

The body of a function is where you write the code that your function runs. It comes directly after your **def** statement and it must be indented.

For example, our **add_num** function might look like this:

```
def add_num(num1, num2):
    total = num1 + num2
    print("The total is: " + total)
```

## Task 3.3: Let's call our function!

Now it's time to call our function!

1. Comment out or delete the `print` statements that you added in Task 1.3 (the same ones that you copied in Task 3.2).

2. Instead of printing, call your `print_board` function here!
   Make sure you pass the `board` variable so it can print out the data.

### *Hint: How do I call my function?*

You can call a function called `add_num` like this:

```
add_num(1, 2)
```

The things inside the brackets are the *parameters* that will get passed through to the function. In this example, `add_num` is being passed two parameters which are both integers (the numbers 1 and 2).

## Task 3.4: Reveal the updated board!

1. At the bottom of your code (after the code from Task 2.4), call the `print_board` function again so that the users can see how the board changes after a move is made!

## ☑ 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!

Now we have the current player's symbol stored in a variable, we can print it out!

1. In your code, after you set the **`symbol`**, `print` out whose turn it is.

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

After Noughts has played their turn, it's time for Crosses to play. At the end of a turn we need to change the **`symbol`** to the other symbol.

1. Add an `if` statement to the bottom of your code that checks to see if the **`symbol`** is Noughts. If it is, change the **`symbol`** to Crosses.

## Task 4.3: Switch back!

Later in the game we'll want to change back from crosses to naughts.

1. Now add code to make it so that if the **`symbol`** was Crosses, it now changes to Noughts.

### *Hint: How do I call my function?*

Make sure you use an `elif` here! Otherwise it'll keep being naughts all the time!

## ☑ 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

While Loops

# Part 5: Wait a while to win?

## Task 5.1: Game Over?

Right now we can only have one turn, which is a bit boring! Until the game is over, we want to keep having turns and making moves.

1. Underneath where the **`board`** list is created, create a variable called **`game_over`**, and set it to `False`.

## Task 5.2: Did I win yet?

Now we will build a while loop and put our game into it so that it only runs while the game is not over.

1. Go to your code just before you print whose turn it is.
2. Add a `while` loop. Make sure to set it to keep running while not **`game_over`**.
3. Indent all of your code after the **`while`** loop, so it will get repeated for each turn of the game.

### *Hint*

You need to make sure you indent the rest of your code that comes after this. The easiest way is to highlight all your code (except your `while` loop line) and push the 'tab' key.

### *Hint:*

When you're stuck in an infinite loop, you can use control + c to quit your program!

## ☑ CHECKPOINT ☑

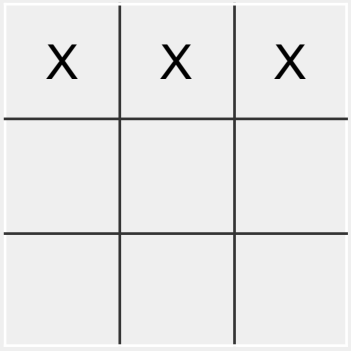
**If you can tick all of these off you can go to Part 6:**

- ☐ 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!

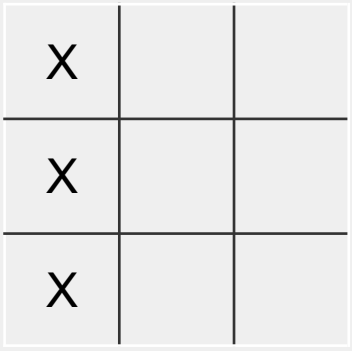
## Task 6.1: Where are the winners

1. Fill in the worksheet below to show all the ways we can win.
2. Write down all the list indexes used in each winning combination.



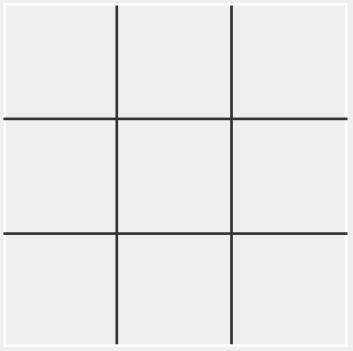

**Indexes used:**

0, 1, 2



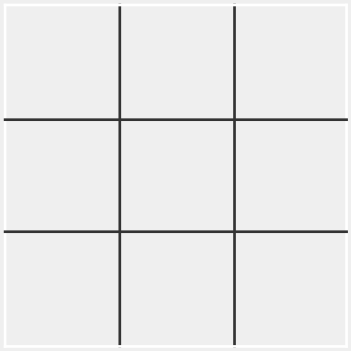

**Indexes used:**

0, 3, 6



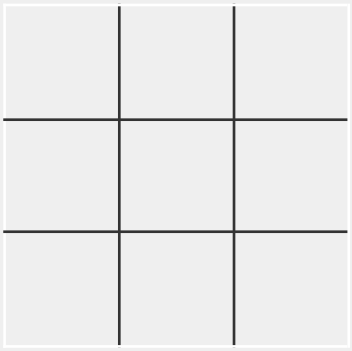
**Indexes used:**

_____



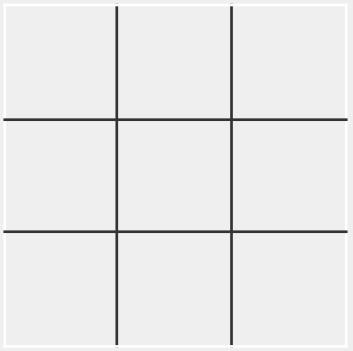
**Indexes used:**

_____



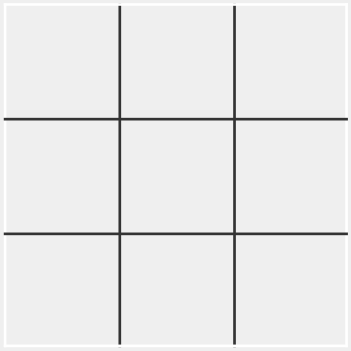
**Indexes used:**

_____



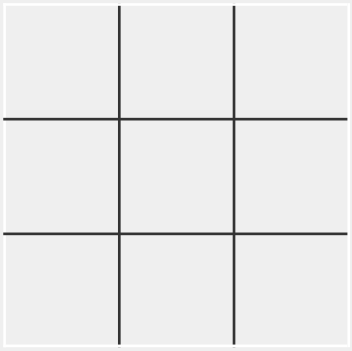
**Indexes used:**

_____



**Indexes used:**

_____



**Indexes used:**

_____

## Task 6.2: Functions again!

Like we built a function to print the board, we will build a function to check for a winner.

1. At the top of your code, define a function called `check_winner`, and pass in the `board` as an argument!

## Task 6.3: One function, two options!

When you write a function, it doesn't matter how you do it as long as you return the right answer at the end!

There's two great ways to work out who is the winner! You get to choose which one you want to use when you write your function.

Here's a description of the two options, **blue** and **orange**! Choose one that you like and follow the steps!

### Option 1: If statements

We'll use `if` statements to check every row, column and diagonal to see if they have a matching set of symbols.

It will be longer code, but a bit faster to write.

**Follow the blue boxes on the next page!**

### Option 2: For loop and lists

We can represent every winning row, column and diagonal as a triple of indexes of the board.

The code will be shorter, but we'll practise using a `for` loop when we write it!

**Skip to the orange box section!**

# ★ Start Option 1 here ★

**Create the body of your `check_winner` function following the blue boxes.**
This function will return `True` if the board has a winning line. And `False` if it doesn't.

## Task 6.4: Checking the top row!

Let's check to see if someone has won on the top row of board! Check all squares on the top row of the **`board`** are the same, and are not blank spaces.

1. Create an `if` statement that checks to see if the 0th, 1st and 2nd items in the list are the same.

2. The symbols might all be the same, but it's not a win if they're blank. Add a check to make sure the symbols are not `" "`.

3. If the game has been won, add a `return True` inside the `if` statement.

### *Hint*

You can chain multiple comparison together:

```
if sister1_bday == sister2_bday == sister3_bday:
    print("We're triplets!")
```

## Task 6.5: Checking the middle row!

Now that we've checked the top row, let's check the middle!

1. Add an `elif` statement under your if statement that checks to see if all of the squares on the middle row of the **`board`** are the same, and are not blank spaces.

2. If they are the same, add a `return True` statement inside the `elif` statement.

## Task 6.6: Checking all the other combinations!

Add more elifs so we can check the last row, the three columns, and the diagonals!

1. Add an `elif` statement to your if statement that checks to see if all of the squares in a row, column or diagonal are the same and are not blank spaces.

2. If they are the same, add a `return True` statement inside the `elif` statement.

## Task 6.7: No winners here!

So we've checked every possible winning combination, but it turns out no one's won yet! Return `False` so the game will keep going.

1. Add an `else` statement to your `if` statement

2. Add a `return False` statement inside the `else` statement.

## ☑ CHECKPOINT ☑

### 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

***You finished Option One! Now you can skip to Part 7***

## ★ Start Option 2 here ★

**Create the body of your `check_winner` function following the orange boxes.**

This function will return `True` if the board has a winning line. And `False` if it doesn't.

### Task 6.4: Winning combinations!

We're going to need a list of all the winning index triples! We'll make this by looking back at where we drew out all the winning combinations by hand.

1. Finish the list of winning combinations:
   Each item in the list is a triple, made up of the 3 square required to win.

```
winning_combos = [(0, 1, 2),
                  (3, 4, 5),
                  ( ,  ,  ),
                  ( ,  ,  ),
                  ( ,  ,  ),
                  ( ,  ,  ),
                  ( ,  ,  ),
                  ( ,  ,  )]
```

**We'll use this list to code the next part!**

### *Hint*

**Did you notice the inner lists have round brackets?**
We call them **tuples**! They're a special kind of list.

You can't add more items to them, and you can't remove existing items from them.
Everyone who sees them knows these triples have to stay as triples!

### Task 6.5: Winning configurations!

We need to store every possible way there is to win.

1. Create a list called `winning_combos` and copy in the list you wrote out in the previous step.

## Task 6.6: Looping over

Now we'll use our list of winning combos one at a time to check the board to see if a winning combo has occurred.

1. Create a `for` loop that loops over every `combo` in `winning_combos`

### *Hint*

Here's an example of a `for` loop:

```
for i in "hello":
    print(i)
```

## Task 6.7: Three squares

Inside the `for` loop we want to use our 3 square combo to calculate the winner.

Since each combo has 3 items, let's store them in 3 separate variables so we can use them!

1. Store the first item in the `combo` in a variable called `combo_part_0`.
2. Store the second item in the `combo` in a variable called `combo_part_1`.
3. Store the third item in the `combo` in a variable called `combo_part_2`.

## Task 6.8: Three symbols

Let's find out what symbols are in those three spots on the board.

1. Look up what symbol is on the board at `combo_part_0`, store it in a variable called `symbol_0`
2. Repeat this for `combo_part_1`
3. Repeat this for `combo_part_2`

## Task 6.9: Winner Winner!

Now that we have the 3 symbols, let's compare them!

1. Create an `if` statement that checks to see if **`symbol_0`**, **`symbol_1`** and **`symbol_2`** are the same.

2. The symbols might all be the same, but it's not a win if they're blank. Add a check to make sure the symbols are not `" "`.

3. If the game has been won, add a `return True` inside the `if` statement.

### *Hint*

You can chain multiple comparison together:

```
if sister1_bday == sister2_bday == sister3_bday:
    print("We're triplets!")
```

You can chain anything you want to compare to everything else: `<`, `>`, `!=`

## Task 6.10: No Winners

If there's no winner, we still need to return a result!

1. At the end of the function, outside the `for` loop, add a `return False` statement.

## ☑ CHECKPOINT ☑

**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

***You finished Option Two! Now you can go to Part 7***

# Part 7: Declare the winner

## Task 7.1: Check whether the game has been won

In the last part, we wrote a `check_winner(board)` function that can tell us when someone has won. **We can use that to tell the user when someone wins!**

1. In the main game loop, after where you call your `print_board` function, call the `check_winner` function.

2. Update the the `game_over` variable with the result that was returned from the `check_winner` function.

## Task 7.2: Declare who won

Now we know that the game won't run forever, we can give a sigh of relief. But at the end of a game, the Tic-Tac-Toe players will want to know who won!

1. Create an `if` statement that checks to see if it is `game_over`.

2. If it is, `print` out who won!

3. Try running the program, and check that the game ends once someone gets three symbols in a row.

## ☑ CHECKPOINT ☑

**If you can tick all of these off you've finished the base game!**

- ☐ The game ends once someone places three symbols in a row
- ☐ Once someone wins, a message is displayed saying who won

Girls' Programming Network

# *Tic-Tac-Toe*

# *Extensions*

*Make your 2 player Tic Tac Toe game even better!*

## This project was created by GPN Australia for GPN sites all around Australia!

**This workbook and related materials were created by tutors at:**

Sydney, Canberra and Perth

Girls' Programming Network

***If you see any of the following tutors don't forget to thank them!!***

### Writers

Renee Noble
Courtney Ross
Anton Black
Mel Bradshaw
Alex McHugh
Lyndsey Thomas

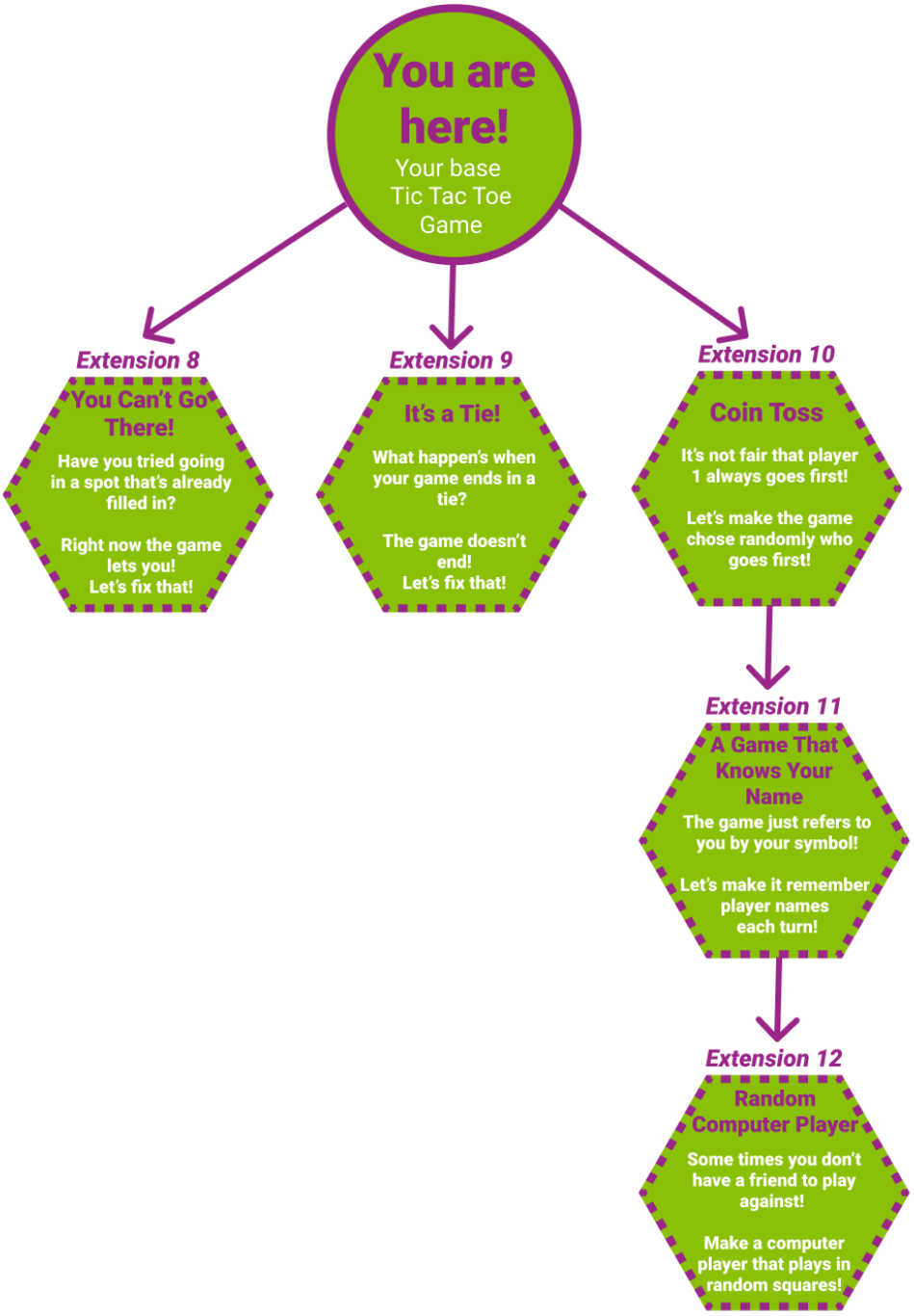
### Testers

Maddie Jones
Jenny Chiem
Debjanee Barua
Kimberley Apted

# What should I do next?

Now you have a two player tic tac toe game you can add these bits to make it even better!

Here are some extensions. You don't have to do them all in order. But some help you do later ones.



# 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?

Your board is represented by indexes show to the right:

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

**We'll create a list in our code shortly. It will store all the squares that are still empty.**

1. List the available squares for the last examples?

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

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

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

free_squares =
[ _____ ]

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

free_squares =
[ _____ ]

## Task 8.2: What spaces are free?

Let's keep track of which places are free. Start by making a list that has all the spaces in it.

1. Go to your start of game code before your `while` loop.
2. Create a list of all the free spaces. It's the start of the game so they are all free.
   `free_squares = [0, 1, 2, 3, 4, 5, 6, 7, 8]`

## Task 8.3: Keeping track of spaces

Now we have a list of all the spaces, we better keep it up to date and remove squares that get played in.

1. Go to the place inside your `while` loop after your print the game board each turn.
2. Remove the space that was just played in from the list of **`free_squares`**.
3. Add a print to print out **`free_squares`**. Play the game and see if they disappear.

### *Hint:*

We can remove an item from a list like this:

```
planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter",
"Saturn", "Uranus", "Neptune", "Pluto"]
planets.remove("Pluto")
```

## Task 8.4: Is that even allowed?

Now we know where we're allowed to go, check to see if the chosen square is allowed!

1. Go to the place in your code just after you have create square_index
2. Check if the chosen move is not allowed!
   Create an if statement that checks if **`square_index`** **is not** `in` **`free_squares.`**
3. If the **`square_index`** is not allowed, print a message saying that square is not allowed.

### *Hint: Checking if a tile position is in the taken_places list*

You can use the `in` keyword to check whether an item is in a list:

```
fruits = ["apple", "banana"]
"orange" in fruits
==> False
```

## Task 8.5: Try that again

We don't want to just tell the player off and then not give them a turn!

1. Under your "not allowed" message add a `continue`.

*This will skip our code back to the start of the loop so they can try again and not lose their turn!*

## ☑ 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

1. Before the while loop, create a variable `counter` to store which turn we're on. Set it to `0`.

2. On each turn, we need to increment counter by 1. Do this inside the while loop just before you call your `check_winner` function.

### *Hint: Increment a counter*

You can increment a variable by adding a value to itself. To increase `my_counter` by 5, we can do the following:

```
mycounter = mycounter + 5
```

## Task 9.2: Check for a tie

1. Add an `elif` statement after you check if the game is over. This `elif` statement should check if the counter is up to nine. If it is the game is a tie.

2. If it's a tie, we should `print` a message so the players know it's a tie.

3. After we print it's a tie add a `break` to escape the loop and end the game.

### *Hint: Breaking out of loop*

Remember how to break out of a loop?

```
while True:
    if my_condition == something:
        break
```

## ☑ 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!

At the top of your code, import the `random` package: `import random`

## Task 10.2: Who will go first?

1. Find the line of code where you decided that one symbol would always go first.
2. Change this line to **randomly** choose `x` or `o`.

### *Hint: Random choice*

You can use `random.choice` to select a random piece of fruit from a list like this:

```
fruit = random.choice(["apple", "orange", "banana"])
```

## Task 10.3: Tell us who's next!

Print the result of the coin toss

1. Make sure you announce which player is going first!

## ☑ 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!

1. Make sure you have done **Bonus 2.5** where you ask for the players names.

## Task 11.2: Who's there

At the start of the game, below where you set `symbol`, also set the current player.

1. Us an if statement to check if player 1's symbol is the current symbol.
   If it is set `symbol` to be `player_o`.

2. Otherwise, the current symbol must be player 2's symbol. Use an else statement, and `symbol` to be `player_x`.

## Task 11.3: It's your turn!

Time to announce the player name!

1. Change the message you print out each turn that says which symbol is up. Change it so is says the `symbol` and the `current_player`.

## Task 11.4: Who's next?

We need to update the `current_player` just like we did with the `symbol`.

1. Go to the place in the code where you use an `if` statement to switch the `symbol` to get ready for the next symbol's turn.

2. Inside those `if-elif` statements, add in code that will also update the `current_player` at the same time.

### *Hint:*

If it was symbol 1's go it's now symbol 2's go and we should change the `current_player` to `p2_name` when we update `symbol`.

## Task 11.5: Who's won?

When someone wins we want to print out their name, not their symbol.

1. Go to the `if-elif` statements where you check for the winner.
2. Change it from printing the winner's symbol to the winner's name.

## ☑ 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 you've done the other extensions you need to do this one:

Have you done...

1. **Extension 8: You Can't Go There**
2. **Extension 11: A Game that Knows Your Name!**

Do them now if you haven't!

## Task 12.2: My name is computer

We want to choose a random move for the computer play from the list of free squares.

We'll need to check if it's the computers turn. If it is, we won't ask for a player to choose a square, we'll pick randomly.

1. Go to your code at the top of your `while` loop inside the loop
2. Use an if statement to check to see if the `current_player` is called ***computer***.
3. If their name is computer then randomly select a `square` from the list of `free_squares`. Set the `square` variable to be this randomly chosen square.

### *Hint:*

We can use random choice like this:

```
import random
fruit_of_the_day = random.choice(["apple", "banana"])
```

## Task 12.3: I'm no robot!

We've handles the computer player now. But our code that asks for a square still happens every time! Let's make it not happen for computer players.

1. Create an `else` statement for your computer checking `if` statement.

2. Inside the `else`, move in the code that asks the user for their square in there. *This code runs whenever it's not a computers turn!*

## ☑ 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 ★

Now that your computer player works, can you figure out how to make it play better than just playing randomly.

Try to figure out how to code up:

- 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

See what you come up with and if you can get it to work! Good luck!