



Girls' Programming Network

Tic-Tac-Toe

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

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
Jess Austin
Jin Cong
Branda Zhong

Testers

Qing Wang
Alex McCulloch
Debjanee Barua
Kay Pengelly
Nicole Bayona
You Kim Chhay
Courtney Ross
Tess Bennetts
Caitlin Bathgate
Julia Szymanski

Part 0: Setting up

Task 0.1: Making a python file

1. Go to <https://replit.com/>
2. Sign up or log in
(we recommend signing in with Google if you have a Google account)

Task 0.2: Making a python file

1. **Create** a new project
2. Select **Python** for the template
3. Name your project **tic_tac_toe**


Task 0.3: You've got a blank space, so write your name!

A main.py file will have been created for you!

1. At the top of the file use a comment to write your name!

Any line starting with # is a comment.

```
# This is a comment
```

2. Run your code using the  **Run** button. It **won't** do anything yet!

✔ CHECKPOINT ✔

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

- ☐ You should have a file called main.py
- ☐ Your file has your name at the top in a comment
- ☐ Run your file and it does nothing!



Part 1: Welcome to Tic-Tac-Toe!

Task 1.1: Welcome!

1. Let's `print` out a message to welcome the user. We can make the computer say anything we want!

For example: `Welcome to Tic-Tac-Toe!`

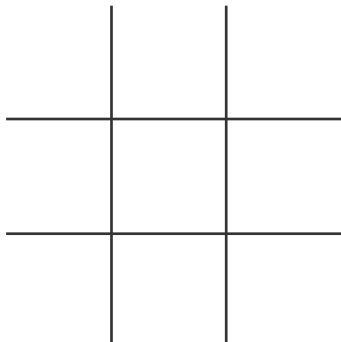
Hint:

Remember to print a message we use:

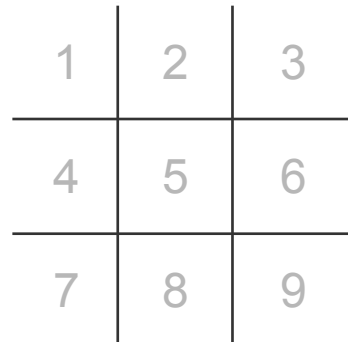
```
print("Hello, World")
```

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**.



Task 1.2: Storing a square

We're going to need to store the symbols that appear on our board.

Let's start by storing what is in the first square on the board. It starts out as a blank space.

1. Create a variable called `square_1` and set it to be a space..

Hint:

You can store a blank space in a set of quote marks!

```
myVariable = " "
```

Task 1.3: Storing more squares

The tic-tac-toe board has 9 squares, so we need to store more squares!

1. Repeat what you did in the last step, but for the rest of the squares, `square_2` to `square_9`

Task 1.4: Printing The Board

Now we want to be able to see the playing board, so we are going to print out each square

1. Use the `print()` function to print 13 dashes (-) for the top border of our playing board.
2. Print the top row of squares. We want to add together some lines " | " and the variables `square_1`, `square_2`, and `square_3`.
3. Now finish off the rest of the board to have 9 squares total!

It should look like this:



Hint:

How do you print out a row of squares and lines? Like this:

```
print("| " + square_1 + " | " + square_2 + " | " + square_3 + " | ")
```

✓ CHECKPOINT ✓

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

- ☐ Print a welcome message for the user.
- ☐ You've created 9 variables each storing a space.
- ☐ You've printed your playing board!

Part 2: Asking questions

Input

Task 2.1: What symbol are you?

We need to ask which player will be playing this turn. Ask which symbol we are up to.

1. Ask which symbols turn it is, store it in a variable called `symbol`.

It should look something like this:

```
>>> Which symbol's turn is it now?
```

You can answer X or O!

Hint

Remember we can ask a question and store the user answer like this:

```
name = input("What's your name? ")
```

Task 2.2: What spot do you want to put it?

Now we need to get the spot on the grid where the player wants to place their symbol for their turn.

1. Ask the player what square they want to play in. Store the answer in a variable called `square`.

Task 2.3: Check what happened!

Let's make sure our variables are storing the player's answers.

1. Add a print statement to print out `symbol` and `square`.
2. Are these the same as what the player typed in?
3. You can delete your extra prints now. They were just to see what was happening.

✓ 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.

★ BONUS 2.4: Hello, who is playing? ★

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

Let's add some more welcomes to the start of the game! Let's also get the names of the players and say hello to them.

Go back to before the welcome message:

1. Ask the user for the name of the first player and store it in a variable called `p1_name`.
2. Do the same thing for the second player, storing it in a variable called `p2_name`.
3. Now we can add the players' names to the welcome message.

Part 3: Marking the board

If
Statements

Task 3.1: Update square 1 using an if statement

Let's check if the player said that they wanted to play in the first square.

1. Create an **if** statement to check if the square the player entered is equal to "1".
2. If it was "1", update `square_1` to be the `symbol` that the user entered

Hint

You might use something like:

```
if square == "1":  
    square_1 = symbol
```

Task 3.2: Now square 2

Let's do it again for square 2!

1. After the **if** statement you wrote in task 3.1 add an **elif** statement to check if the the square the player entered is equal to "2".
2. Inside that **elif** update the `symbol` in `square_2`.

Task 3.3: Repeat repeat repeat

Let's do it again and again for the other 7 squares!

1. Repeat the **elif** you did in Task 3.2 to check the numbers 3 - 9 and update the correct squares.

Task 3.4: Let's print out the board again

1. Reuse the code from Part 1.3 to **print** out the board. Place this after your **if-elif** statements!

Just copy-paste the code from Part 1.3!

✓ CHECKPOINT ✓

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

- ☐ We have an if statement and 8 elif statements to update the square variables with symbols.
- ☐ We have code that prints out the board following the if-elif statements.
- ☐ The board is updated correctly with the player's move.

★ BONUS 3.5: What about square 10? ★

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

What if they enter something that isn't the numbers 1-9? Let's print out that they made a mistake!

1. After your `if-elif` add an `else` statement. This will catch anything that is not the numbers 1 - 9.
2. Print out a message that tells them they made a mistake.
Like `"You can't go there!"`

Part 4 : Let's Play!

While Loops

Task 4.1: Taking many turns!

So far our game only has one turn. Let's add a loop to make our code repeat so there will be more turns!

1. Create a **while** loop that will run forever. Add this before you ask for the players symbol!

Hint:

To create a **while** loop we can set it to be **True** to run the code inside forever!

```
while True:
    # Some cool code!
```

Task 4.2: Add your code!

We need to move all our code that happens every turn inside the body of the loop. We do this by indenting it.

1. Indent all your code that is under the **while** loop line.

Task 4.3: Run your code!

Let's make sure the **while** loop works!

1. Run your code. Does it let you have multiple turns?

Hint:

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

☑ CHECKPOINT ☑

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

- ☐ The game ask for a symbol and a square and prints them out on the board.
- ☐ The game repeats the previous checkpoint over and over.

Part 5: Picking the winner

Complex
logic

Task 5.1: Let's figure out all the ways we can win

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

X	X	X

Variables used:
square_1, square_2,
square_3

X		
X		
X		

Variables used:
square_1, square_4,
square_7

Variables used:

Variables used:

Variables used:

Variables used:

Variables used:

Variables used:

Task 5.2: Check the top row is the same.

After you've printed the board out with the new move on it, it's time to check to see if the symbol that was just placed made the player win the game!

1. Go to the place in your code after you've printed the board out with the new symbol on it. Make sure you are still indented inside the `while` loop.
2. Let's use an `if` statement to compare all the squares on the top row. Check to see if `square_1`, `square_2` and `square_3` are all equal to each other.
3. Now if we know it is a winning move it's time to announce the winner. Inside the body of the `if` statement `print` out the symbol that is in one of the `square_1`, `square_2` and `square_3`, it's the winning symbol!
4. One last thing. We want the game to end now that we found a winner. After printing the winner add a `break` statement to break the loop and end the game.

Hint:

Check if variables are equal by using two equal signs. We can check if two, three or more variables are equal to each other by using `and`.

```
if variable1 == variable2 and variable2 == variable3:  
    print("yay!")
```

Task 5.3: Let's test our code!

1. Test your code with following three possibilities:
 - a. Try putting all the same symbols on the top row.
 - b. Try putting different symbols in the top row.
 - c. Try not to put any symbols on the top row.
What happens? Did " " win?

It's not enough that they are all equal. We need to make sure they are not a blank space.

2. **Fix** the `if` statement!
Add a check to make sure the squares are **not equal** to " ".

Hint:

We can add "not equals" to our chain too!

Here there first 2 dinners are equal, but not the third:

```
if dinner_1 == dinner_2 and dinner_1 != dinner_3:  
    print("I had the same thing for dinner, but only twice")
```

Task 5.4: Code for all winning scenarios

We now need to code for the other 7 winning combinations.

1. Create 7 `elif` statements to go with your `if` statement from the previous steps. These 7 `elif`s should test all the other possible winning combinations.
2. Make sure to `print` the winner and `break` out of the game loop in each `if` and `elif`.

✓ CHECKPOINT ✓

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

- ☐ You have `if` or `elif` statements for all winning combinations
- ☐ The game should print out the winner and finish for all 8 winning combinations



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
Jess Austin
Jin Cong
Branda Zhong

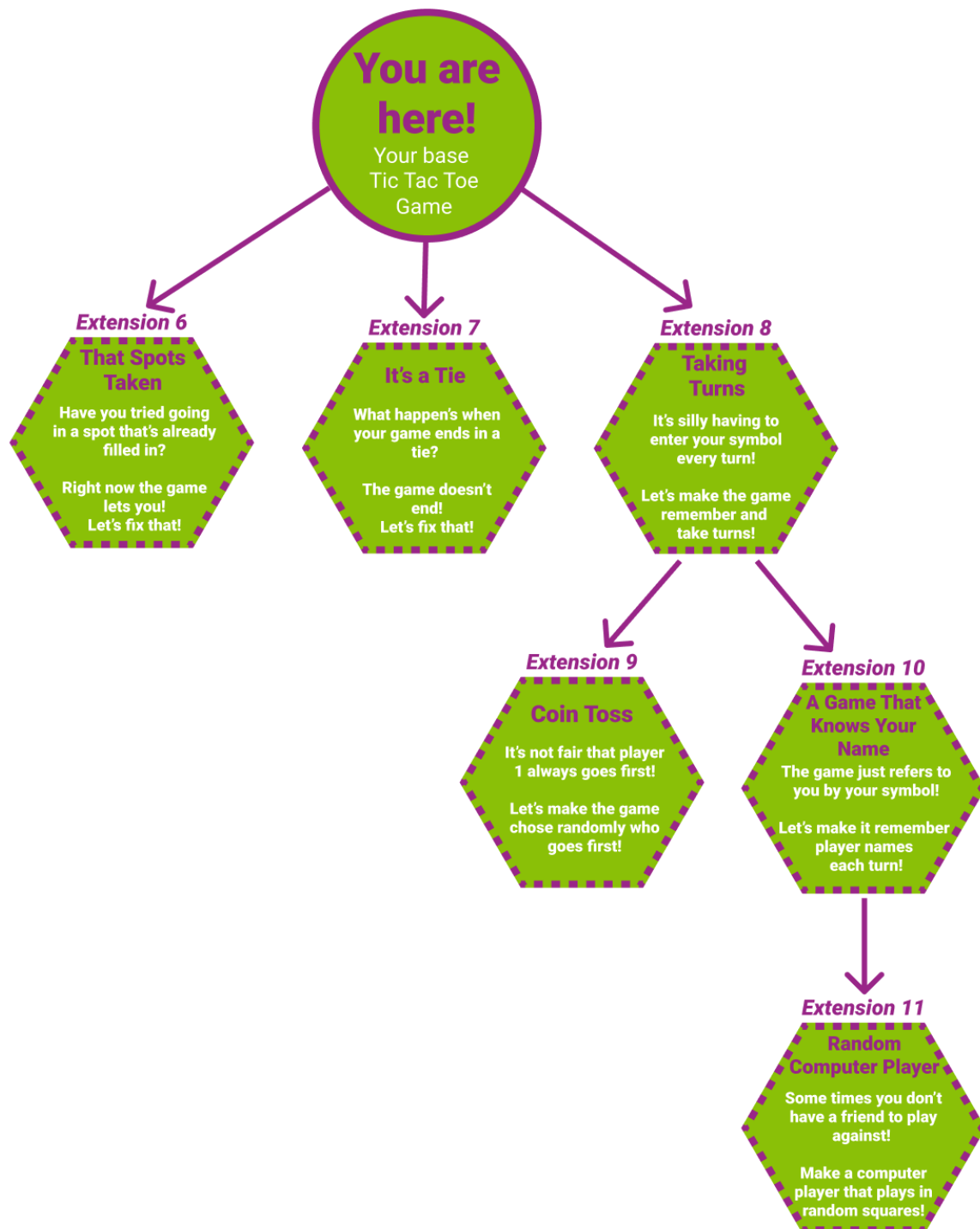
Testers

Qing Wang
Alex McCulloch
Debjanee Barua
Kay Pengelly
Nicole Bayona
You Kim Chhay
Courtney Ross

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.



6. Extension: That spot's taken!

At the moment the game lets you go in a spot someone else has already taken!

Let's make sure no one can cheat by adding a checker to see if the position is already filled in.

Task 6.1: That's not an empty spot!

Until now we just let the player say which square they want and then we went there.

Let's add a check to make sure that spot doesn't already have a symbol.

1. Go to your code and find the section with 9 `if-elif` statements for updating the correct variable with a symbol.
2. Change your first if statement to make sure it's still free, add a check to your if statement condition to make sure `square_1` is still a blank space " ".

Hint

You can check for two things using `and`:

```
birth_month = "march"
birth_day = 16
if birth_month == "march" and birth_day == 16:
    print("We have the same birthday!")
```

Task 6.2: Repeat!

Now we end to add that check to the code we use for updating all the other squares!

1. Repeat the last task for your 8 other square elif statements.
2. Test your code by playing the game and seeing if the game now stops players for cheating by just not placing their

Task 6.3: Access denied!

If you did **Bonus 3.5** about adding an *else* to the end of your *if-elifs* you don't need to do this part. But you can update your message.

1. After your *if-elif* add an *else* statement. This will catch anything that is not the numbers 1 - 9.
2. Print out a message that tells them they made a mistake.
Like "You can't go there!"

✓ CHECKPOINT ✓

If you can tick all of these off you've finished Extension 6:

- ☐ If you try to play in a square that is already taken the game **doesn't** overwrite the symbol that's already there.
- ☐ The game prints a message telling you that spot is already taken if you try to play in a filled square.

7. Extension: It's a tie

At the moment the game only tells us if someone wins. Let's add a message for when the players tie.

Task 7.1: Count your moves

Let's keep track of how many moves have been made.

1. Before the game starts, create a variable `counter` and set it to zero.

Task 7.2: Add one

At the end of each turn, we need to increase the counter!

1. Before your code that checks for a winner, increment the value of `counter` by 1.

Hint

We can update the value of a variable by adding a new value to itself

```
my_number = my_number + 5
```

will increase `my_number` by 5

Task 7.3: Check for a tie

If we get to 9 moves and do not have a winner, we can say the game is a tie.

1. After all our `if-elif` statements for checking for a winner, let's add an `if` statement to check if we have reached 9 moves. If this is `true`, it's a tie and we can `print` a message to let the players know.
2. Also add a `break` statement inside your `if` statement, so that the game ends.

Task 7.4: Players make mistakes!

Sometimes players make mistakes! Like putting in a square that doesn't exist like 99 or like trying to play in a square that is already filled in.

But what if a player makes a mistake! That turn shouldn't count!

1. Run your code and try to enter a square that doesn't exist. Like 99.
2. Run it again and try playing in a place that already has a symbol in it.

You might have added code to stop players from cheating or breaking the game already! We did this in *Bonus 3.5* and *Extension 6*.

Let's not move add onto the counter if that happens. Let's make your program skip back to the start of the loop, to stop the game from switching players .

3. Go to the **else** statement part of your **if-elif-else** that updates the square variable with the current symbol.
4. Inside the else statement, at the end, add a **continue**.

✓ CHECKPOINT ✓

If you can tick all of these off you've finished Extension 7:

- ☐ You should have a variable that keeps track of how many moves have been made.
- ☐ Your counter should increase by 1 each time a player makes a move
- ☐ The game ends if 9 moves have been made and no-one has won.

8. Extension: Taking Turns

It's silly to have to enter which symbol you are every turn! The computer should remember and switch who's turn it is by self!

Add an extension to your program so it asks each player which symbol they are at the start of the game and then remembers. The program should alternate whose go it is every turn and tell the player which symbol's turn it is.

Task 8.1: Removing old code!

We don't want to ask what symbol is happening every turn. Get rid of that code!

1. Remove the line inside the loop which asks which symbol we are.

```
symbol = input("What symbol are you? ")
```

Task 8.2: Only ask once!

At the start of the game before we start playing we want to ask player 1 and player 2 what symbols they are.

1. Go to the place in you code just before your **while** loop
2. Ask the user which symbol player 1 is using. Store it in a variable called **symbol_1**.
3. Next, ask what symbol player 2's will be using, and store it in a variable called **symbol_2**.
4. After finding out the two symbols, create a variable called **symbol** and set it to start as **symbol_1**.

Hint:

You could ask for player one's symbol like this:

```
symbol_1 = input("What symbol are you player 1? ")
```

Task 8.3: It's your turn!

We want to announce which symbol we are up to every turn!

1. In your code go to the start of the `while` loop, we'll add the following code inside the loop.
2. Add a `print` statement that tells the users which is the `symbol`.

For example: `It's X's turn!`

Task 8.4: Change Symbols!

Every turn, the final thing we want to do in the the loop is to swap switch what symbol is the current symbol. This will get us ready for the next turn!

1. Go to the bottom of your while loop, make sure it is still indented inside the loop
2. Use an `if` statement to check if `symbol` is equal to `symbol_1`. If it is, change `symbol` to `symbol_2`.
3. Add an `elif`, so we can swap the symbols back from `symbol_2` to `symbol_1` on the next turn!

Task 8.5: Players make mistakes!

Sometimes players make mistakes! Like putting in a square that doesn't exist like 99 or like trying to play in a square that is already filled in.

But what if a player makes a mistake! They shouldn't miss their turn!

5. Run your code and try and enter a square that doesn't exist. Like 99.
6. Run it again and try playing in a place that already has a symbol in it.

You might have added code to stop players from cheating or breaking the game already! We did this in *Bonus 3.5* and *Extension 6*.

If you've done Extension 7, you've already solved this problem!

Let's not move onto the next layer just yet. Let's make your program skip back to the start of the loop, to stop the game from switching players .

7. Go to the `else` statement part of your `if-elif-else` that updates the square variable with the current symbol.
8. Inside the `else` statement, at the end, add a `continue`.

☑ CHECKPOINT ☑

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

- ☐ You store both players' symbols in variables.
- ☐ You have selected a player's symbol to go first.
- ☐ At the end of each turn, the current symbols swaps and the other player gets a turn.
- ☐ You have tested your code to make sure the board is always updated with the right symbol.
- ☐ A player doesn't lose their turn for making a mistake.

9. Extension: Coin Toss!

It's not fair that player one always gets to go first! Let's fix that!

Task 9.1: Prepare yourself!

1. Make sure you have done **Extension 8** about taking turns!

Random

Task 9.2: Import random

We need help making random choices, let's import the library that does that!

1. At the top of your code, add the following statement:

```
import random
```

Task 9.3: Randomise who goes first

We used to just decide that one symbol always went first! When we set `symbol` to be `symbol_1`.

But to be fair we need to randomly choose between the two players symbols to choose a starting symbol.

Update your code where you set `symbol`. Set it to a random choice of `symbol_1` and `symbol_2`

Hint:

We can use random choice like this:

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

Hint:

You're choosing between the variables `symbol_1` and `symbol_2` put them in your list!

✓ CHECKPOINT ✓

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

- ☐ The game randomly chooses who goes first and prints it out

10. Extension:

A game that knows your name!

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

Task 10.1: Prepare yourself!

1. Make sure you have done **Bonus 2.4** where you ask for the players names.
2. Make sure you have done **Extension 8** about taking turns!

Task 10.2: Who's there

At the start of the game, below where you set `symbol1`, also set the current player.

1. Use an if statement to check if player 1's symbol is the current symbol. If it is, set `current_player` to be `p1_name`.
2. Otherwise, the current symbol must be player 2's symbol. Use an else statement, and `current_player` to be `p2_name`.

Task 10.3: It's your turn!

Time to announce the player's name!

1. Change the message you print out each turn that says which symbol is up. Change it so it says the `symbol1` and the `current_player`.

Task 10.4: Who's next?

We need to update the `current_player` just like we did with the `symbol1`.

1. Go to the place in the code where you use an `if` statement to switch the `symbol1` 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 `symbol1`.

Task 10.5: 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 10:

- ☐ 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.

11. Extension: 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 a 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 each turn. (I hope none of your friends' names are Computer!)

Task 11.1: Prepare yourself!

1. If you haven't already done bonus **Extension 10** about storing the players name, do it now!

Task 11.2: What spaces are free?

The computer will need a list of moves it's allowed to randomly choose from. Let's make that list.

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 = ["1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

Task 11.3: 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 11.4: My name is computer

Now we need to choose a move for the computer!

We'll need to check if it's the computer's turn. If it is, we won't ask for a player to choose a square, we'll pick randomly. Players will still get to pick a square on their turn though!

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:

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

Task 11.5: I'm no robot!

We've handled 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 computer's turn!*

☑ CHECKPOINT ☑

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

- ☐ If you say a computer is playing the game randomly chooses moves for the computer's 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.