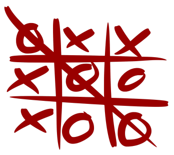
Tic Tac Toe Extensions!



Extension 1: Choose your symbol!

Now that we've mastered naughts and crosses, why don't we make the board look more interesting by letting the users pick what their symbol will be?

Let your imagination run wild! Your symbol could be ":P", ":)", "*_*", or anything you can think of. Try and make sure your users can't both pick the same symbol though!

Task E1.1:
Some things you'll need to use for this

- `input`
- Variables to store each player's symbol

Extension 2: Multiple games and scores

In real life, you can play as many rounds as you like of Tic Tac Toe. It's exactly the same with the computer!

Ask the player how many games they want to play (store this integer in a variable called `total_games`). You should also make a different variable to keep track of how many games you've played. The computer should also keep track of who wins each round to decide who wins the game.

Task E2.1:

1. At the beginning of your code, make a new variable called `total_games` and ask the player for the number of games they would like to play.

2. Now make a variable called `games_played` and think of a way you could use this in your code to count the number of games played.

3. Make two variables (`player0_score` and `playerX_score`) which keep track of how many games each player has won.

4. Add some code so that your game runs until the number of `games_played` is the same as the number of `total_games`. *(Hint: You'll need another while loop or use a for loop)*

5. After each round, add one score to the winning player's score.

6. When the game ends, print the scores of each player so you can see who won the most rounds!

Extension 3: Turns

We've already learnt how to change the turn, but can we do it based on which turn number it is?

`player0` always starts first, which means if the turn is an *even number*, it's `player0`'s turn to pick a square. Try using the **modulo (`%`)** operator to see whose turn it is!

Task E3.1:

1. Check whose turn it is and tell that player it's their turn.

2. Remember to change the current symbol!

Extension 4: Print function

We could print out the game board by writing it out every time, but this would take a while, so let's think of a different way!

By writing a function that tells the computer how to print the game board, we can then tell the computer to print it for us without repeating ourselves. We can call this function any time we need to see the game board.

Task 4.1:
Write a function `board_layout` that prints out the game board.

You'll need to use:
- `def`
- `grid`

Extension 5: A loopier win!

Before, we checked to see if a player won by using `if` and `elif` statements, or by using one `if` statement and using `and` and `or`! But there's another way.

We can use a for loop and range to check the rows and columns to see if a player has won. Remember to look at how your board is numbered!

A reason we might prefer this is if we want to be able to adjust the board size later on. Perhaps we wish we could change this game to make Connect 4, we don't want to have to write the whole thing over again.

Task E5.1:

1. Use a `for` loop and `range` to check to see if a player won across a row.
2. Use a `for` loop and `range` to check to see if a player won across a column.
3. Write a function `check_win`, and place all your code that checks for a win in there.

Extension 6: A silly opponent!

To do this ask your tutors how to use random!

Sometimes we don't have a friend with us to play against! So let's make a computer player! It, just might not be very smart!

We're going to make a computer player that chooses a random spot on the board to play!!

We need to edit our existing game and add some bits to keep track of the game!

Task E6.1:

1. Copy all of your code to a new file! Call it silly.py !
2. At the start of the game create another list that stores the number 1 to 9! This is going to be a list of all the squares that are still available on the board. So name it `available_squares`
3. Go back through your code, find where a player places a X or O on the board. Remember `input_square`, it stores the number square we want to play in. After we play a symbol on the board, `remove input_square` from `available_squares`
4. Instead of asking for `input` for the second player's name, set the second player's name to `"Computer"` in the code.

Hint: do you remember how to use `my_list.remove("item")` *from the lectures?*

Now we need to teach the computer how to take turns! The computer is always going to be crosses (X)!

Task E6.2:

1. `If` the `current_symbol` is naughts, we're going to ask the user for input (like we've been doing all along). *Only put the part up until after your input gettign while loop in this if statement.*
2. But if the `current_symbol` is X, we're going to do something `else`
3. Inside that else statement, print out `"Computer player thinking"`

If you run it now, your computer player will think for ever!

It's silly time. We're going to teach the computer to make silly decisions using `random`

Task E6.3:

1. `import random` at the top of the file
2. Where your computer player is thinking, use `random.choice(available_squares)` to chose a random square to assign to the variable `input_sqaure`

You just taught the computer how to choose a random input. The rest of the game is the same!

Artificial Intelligence

Now imagine how cool it would be if we played the game against the computer. Yes!!! So now let's make some modifications in the code so that the computer can decide by itself when making a move.
We're going to teach the computer how to check if there's a move that wins the game. If there is it will play there! Let's get started!

Task 1:

Let's first assume that the computer wants to place the character in square one. Can you think of all the states the computer might win? Yes! There are three possibilities. Try writing a code that checks these possibilities and if any of them is true, announces the computer as the `winning_player`.
You'll need to use:

- `if` statements
- `grid`
- `Available_squares` (from Extension 7)
- `winning_player`

Task 2:

Now let's take a further step and expand what you did in Task 1 to check if there's a winning move if computer places the character in ***any*** of the squares on the board.
You'll need to use:

- `if` and `elif` statements
- `grid`
- `available_squares`
- `winning_player`

Task 3:

But what if there aren't any winning moves and it's still computer's turn. How should the computer decide then?

Tell the computer to choose the first possible move out of the middle (square 5), a corner (squares 1, 3, 7, 9), and an edge (squares 2, 4, 6, 8). Remember we want to make the computer to do the thinking.

Artificial Intelligence 2.0

Well done! You have now made the computer smarter! But you're probably thinking how long and messy your code looks now. So let's make your code neater and easier to read.

We can do this by putting some of the code, such as the computer's move, into another python file. We can then import this code back into our game to use!

Task 4:

Create a new python file in the same folder as your game and call it *`computer_move.py`*
You can take a look back at **Part 0** of the workbook.

Task 5:

In *`computer_move.py`* file, create a function and call it *`get_computer_move`*. Then copy all your code from **Part 10** and paste it this function.

Hint: We already created a function in Extension 5!

Task 6:

Now you want to use the *`get_computer_move`* function in your code, but since it's not in the same file as your code you have to import it first.

So how do we import? Just add the following line:

- `from` computer_move `import` get_computer_move

***Note:** computer_move.py and tictactoe.py files must be in the same folder so that you can import one file into the other.*

Task 7:

Now that you've imported the function, we need to use it! Remove all of your code from **Part 10** and replace it with your function name.