

Lab1 (Sep 14)

This lab is on Python programming.

Academic honesty and standard:

This is an **individual** assignment. The work you submit must be your own work. Obviously you should be able to fully explain and describe possible alternatives for any line of code you include in your submission. Do not share any part of your code or your design thinking. Remind yourself of the UBC policies on Academic honesty and standard.

Submissions:

CPEN333B: You will submit one python3 file (.py extension) for the implementation of part1 only. Submit the file to the associated Canvas assignment dropbox by the deadline.

CPEN333A: You will submit two python3 files (.py extension), one for the implementation of part1 and one for the implementation of part2. Make sure to clearly include the words part1 or part2 in the name of your submitted files. Submit the files to the associated Canvas assignment dropbox by the deadline.

Do not forget to include your name and student number as comments at the beginning of each .py file.

Due: **Thu Sep 14, by 13:58 PM** (common to everyone)

Please plan ahead, no late submissions are accepted.

Write the best code you can. For marking, we will test for functionality first (must work), and we will check the code design, readability, documentation, ..



Lab room usage rule for Lab 1:

- Make sure **you can login to the lab computers** (this is not related to lab1 deliverable). This is needed for our next lab.
- TAs will be in the lab during your scheduled lab time to help you, and I do recommend that you use the lab time wisely, however, for Lab 1 we will not record attendance and there is no scheduled demo for it.

Prep:

- The lab computers have Anaconda and Visual Studio Code installed. If you are using the lab computers, make sure you do NOT leave your code/work in a publicly accessible folder.
- For your own machine:

- Make sure you have a current python 3 installed on your machine. Using the [Anaconda installation \(https://docs.anaconda.com/anaconda/install/index.html\)](https://docs.anaconda.com/anaconda/install/index.html) will not only install python, but also some really useful tools such as package manager, Jupyter notebooks, spydr, ...
- Install an IDE. [Visual Studio Code \(https://code.visualstudio.com/download\)](https://code.visualstudio.com/download) is a great choice. In VS code, install its [python extension \(https://code.visualstudio.com/docs/python/python-tutorial\)](https://code.visualstudio.com/docs/python/python-tutorial) (by Microsoft).
- Start early.

Make sure your python is current. We consider the latest stable version of python that anaconda uses as current (unless specifically stated otherwise). You can update your anaconda installation by following instructions here: <https://docs.anaconda.com/free/anaconda/install/update-version/>  and <https://www.anaconda.com/blog/keeping-anaconda-date>  <https://www.anaconda.com/blog/keeping-anaconda-date>.

Implementing Tic-Tac-Toe game

Part 1: In this lab, you will implement a command-line tic-tac-toe game (<https://en.wikipedia.org/wiki/Tic-tac-toe> (<https://en.wikipedia.org/wiki/Tic-tac-toe>)). The game is one player against the computer, and for simplification for this part, the computer makes valid moves randomly not intelligently. You can use the [randint \(https://docs.python.org/3/library/random.html#random.randint\)](https://docs.python.org/3/library/random.html#random.randint) method in the random module.

Watch this video as a demo for the game first (your implementation should closely match the demoed program):

<https://youtu.be/jBhbtZjYhCM>  (<https://youtu.be/jBhbtZjYhCM>)



<https://youtu.be/jBhbtZjYhCM>

As mentioned before, follow the specification and print the messages and the board exactly as you see in the demo video and screenshots.

Consider the following code template.

The `init` method is given (use as is) as well as the main superloop code for running the game (use as is). You are to use the two variables `board` and `played` correctly in your code (they are to be updated after every move during the game, and MUST be current at any state of the game).

Do not add any new functions, variables with global scope, or types. Do not change the function headers or the structure of the program. We may use automated tests as a part of grading, so every unit (e.g. functions or methods) must be testable fully on own. You are allowed to use inner functions if needed (as a background: C does not allow nested functions, but python's inner function is similar to local functions in C# or Java).

See the code for the comments. You should add useful comments for the code you add.

You are to complete the requested functions in the program and submit your code as a `.py` file.

```
# student name:
# student number:

# A command-line Tic-Tac-Toe game
import random

board = [' '] * 9 # A list of 9 strings, one for each cell,
                  # will contain ' ' or 'X' or 'O'
played = set()    # A set to keep track of the played cells

def init() -> None:
    """ prints the banner messages
        and prints the intial board on the screen
    """
    print("Welcome to Tic-Tac-Toe!")
    print("You play X (first move) and computer plays O.")
    print("Computer plays randomly, not strategically.")
    printBoard()

def printBoard() -> None:
    """ prints the board on the screen based on the values in the board list """
    pass #To Implement

def playerNextMove() -> None:
    """ prompts the player for a valid cell number,
        and prints the info and the updated board;
        error checks that the input is a valid cell number
    """
    pass #To Implement

def computerNextMove() -> None:
    """ Computer randomly chooses a valid cell,
        and prints the info and the updated board
    """
    pass #To Implement

def hasWon(who: str) -> bool:
    """ returns True if who (being passed 'X' or 'O') has won, False otherwise """
    pass #To Implement

def terminate(who: str) -> bool:
    """ returns True if who (being passed 'X' or 'O') has won or if it's a draw, False otherwise;
        it also prints the final messages:
            "You won! Thanks for playing." or
            "You lost! Thanks for playing." or
            "A draw! Thanks for playing."
    """
    pass #To Implement

if __name__ == "__main__":
    # Use as is.
    init()
```

```

while True:
    playerNextMove()          # X starts first
    if(terminate('X')): break # if X won or a draw, print message and terminate
    computerNextMove()        # computer plays O
    if(terminate('O')): break # if O won or a draw, print message and terminate

```

At the beginning of the game the following banner is printed and then the game starts by allowing the player to state a move. The player is X (first move) and the computer is O.

```

Welcome to Tic-Tac-Toe!
You play X (first move) and computer plays O.
Computer plays randomly, not strategically.

```

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

```
> Next move for X (state a valid cell num):
```

The move is stated by the number of a valid cell. If the number is not valid (already taken, or not between 0 and 8) or if it is not a number, the game should display the error message (as shown) and repeat asking for the next move.

```

> Next move for X (state a valid cell num): x
Must be an integer
> Next move for X (state a valid cell num): 13
Must enter a valid cell number
> Next move for X (state a valid cell num): 4
You chose cell 4

```

0	1	2
3	4	5
6	7	8

0	1	2
X		

Computer chose cell 5

0	1	2
3	4	5
6	7	8

0	1	2
X		

```

> Next move for X (state a valid cell num): 

```

In the above image, the computer immediately made a move by choosing a valid cell number randomly and then it displays the updated board.

If we choose a cell number that is already taken, an error message "Must enter a valid cell number" is printed and it will again prompts for an input.

```
> Next move for X (state a valid cell num): 4
Must enter a valid cell number
> Next move for X (state a valid cell num): 0
You chose cell 0
```

X	0 1 2
---+---+---	---+---+---
X 0	3 4 5
---+---+---	---+---+---
	6 7 8

Computer chose cell 7

X	0 1 2
---+---+---	---+---+---
X 0	3 4 5
---+---+---	---+---+---
0	6 7 8

```
> Next move for X (state a valid cell num):
```

The game continues until either the player or the computer wins, or it is a draw. The game prints a final message: either of "You won! Thanks for playing." or "You lost! Thanks for playing." or "A draw! Thanks for playing." messages, and it terminates.



```
> Next move for X (state a valid cell num): 8
You chose cell 8
```

X	0 1 2
---+---+---	---+---+---
X 0	3 4 5
---+---+---	---+---+---
0 X	6 7 8

You won! Thanks for playing.

Part2: [see the Submissions section at the top of this page on whether you need to do part2 as well or not]

Keep the structure of the program from part1 as much as possible as is, but modify the functionality so that you implement the following two features:

- The program uses a random draw (50-50) at the beginning to decide whether the gamer starts first or the computer starts first. Like before we assume that the first one to play plays X.
- The computer plays strategically to win (contrary to part1 for which it chose moves randomly). For example, implement strategies explained here <https://www.rd.com/article/how-to-win-tic-tac-toe/>  <https://www.rd.com/article/how-to-win-tic-tac-toe/#:~:text=When%20you%27re%20the%20first,O%27%20in%20the%20center%20box> or <https://www.wikihow.com/Win-at-Tic-Tac-Toe> 

Fully document your implementation by clearly including comments for any modification to the program structure, any addition, as well your explanation for your implemented strategies. This would include some design decisions.

Do not use any new imports. We do not use user-defined classes in this lab.

To get full mark:

- Your program must work. There won't be any credit for a code that does not.
- Your program must be correct. Fully test it.
- Your program must follow the specification (implement what is requested). Your program must not do any unnecessarily or repetitive work.
- Your program must use good code style, be readable, and include "useful" comments statements.

See the submission dropbox for the marking detail.