How to Use *Torii*

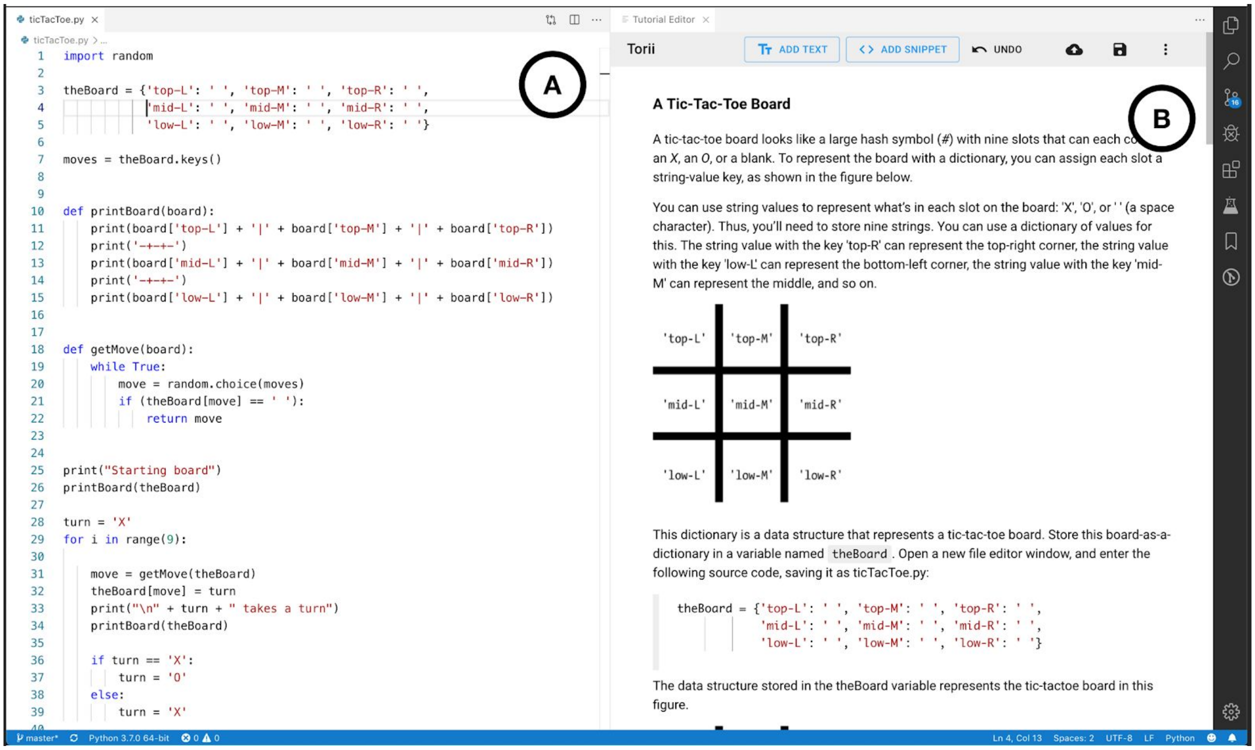
Purpose

In this study, you will use a tool called **Torii** for in order to create and make updates to tutorials. Torii is a prototype tool for writing tutorials with features to help **keep the code and outputs in the tutorial consistent**, and to help you **check the code is complete**.

Follow this guide to get familiar with Torii. Please ask the facilitator any questions you might have about how the tool works and creates outputs. This study is about understanding the usability and expectations about the tool, and this information is important to us.

Step 1: Layout of Torii

The experimenter will open Torii for you on a pre-loaded tutorial.



```python
import random

theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

moves = theBoard.keys()


def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])


def getMove(board):
    while True:
        move = random.choice(moves)
        if (theBoard[move] == ' '):
            return move


print("Starting board")
printBoard(theBoard)

turn = 'X'
for i in range(9):

    move = getMove(theBoard)
    theBoard[move] = turn
    print("\n" + turn + " takes a turn")
    printBoard(theBoard)

    if turn == 'X':
        turn = 'O'
    else:
        turn = 'X'
```

A Tic-Tac-Toe Board

A tic-tac-toe board looks like a large hash symbol (#) with nine slots that can each co an *X*, an *O*, or a blank. To represent the board with a dictionary, you can assign each slot a string-value key, as shown in the figure below.

You can use string values to represent what's in each slot on the board: 'X', 'O', or ' ' (a space character). Thus, you'll need to store nine strings. You can use a dictionary of values for this. The string value with the key 'top-R' can represent the top-right corner, the string value with the key 'low-L' can represent the bottom-left corner, the string value with the key 'mid-M' can represent the middle, and so on.

This dictionary is a data structure that represents a tic-tac-toe board. Store this board-as-a-dictionary in a variable named `theBoard`. Open a new file editor window, and enter the following source code, saving it as ticTacToe.py:

```python
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

The data structure stored in the theBoard variable represents the tic-tactoe board in this figure.

Torii contains **two windows**. On the left (A), the view of the **source code** you're trying to make a tutorial from. On the right (B), an interactive tutorial editor.

(continue on next page)

Step 2: Change the Code

In Torii, **code is linked** between all code-related artifacts:

- Changes to the source code affect the snippets in the tutorial
- Changes to the snippets in the tutorial affect the source code
- The outputs update when the code is changed.

In this step, you should:

- **Make some changes to the source code**, and see it change the snippets
- **Make some changes to the snippets**, and see it change the source code

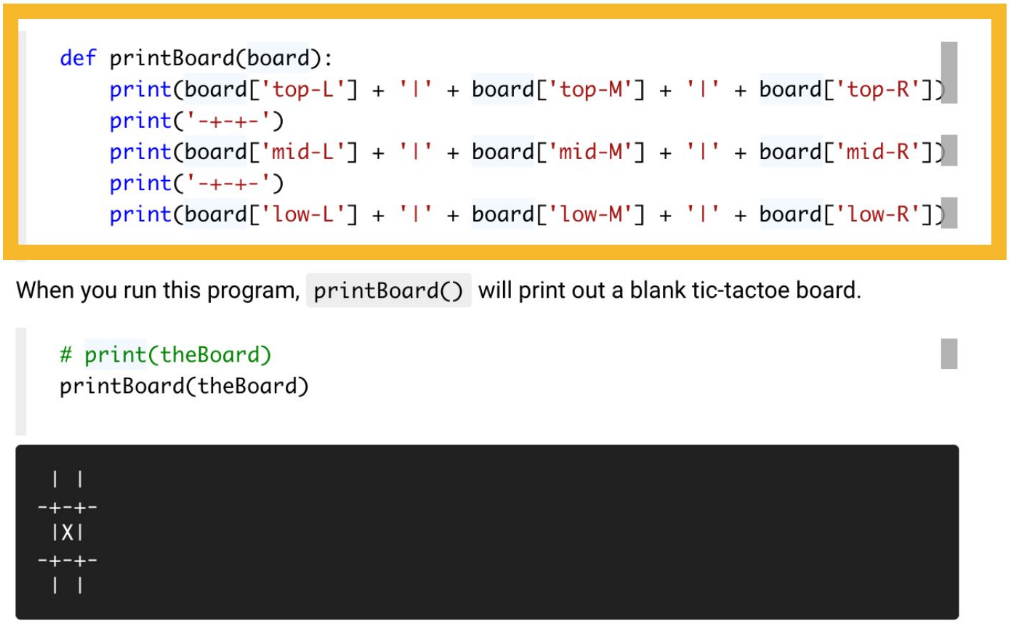
(*Note*: Sometimes, changes you make to the code will *not* be mirrored in the other places the code appears . This is because there can be versions of snippets, which we'll discuss later).

(continued on next page)

Step 3: Outputs Get Updated Automatically

In Torii, the outputs are linked to the code that generated them, and they update live.

Scroll down in the tutorial pane until you see this snippet:



```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
```

When you run this program, `printBoard()` will print out a blank tic-tactoe board.

```
# print(theBoard)
printBoard(theBoard)
```

```
 | |
-+-+-
 |X|
-+-+-
 | |
```

Change some code in the snippet and see the outputs update.

If you want inspiration, try changing all of the “+” signs in the second and fourth print statements to “x”s. You should see the printed output update to have “x”s.

(continued on next page)

Step 4: Outputs Shows a Reader’s View of the Code *So Far*

Outputs show the output of all of the code that has appeared in the tutorial *so far*.

Move an output up and down in the tutorial, above and below other snippets in the tutorial. See how its output changes based on its position.

You can move an output by click and dragging it.

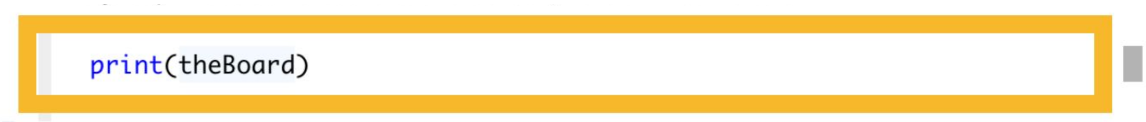
(continued on next page)

Step 5: What Do the Outputs *Really* Show?

Unlike Jupyter notebook, or a Python terminal, Torii does **NOT** run code in the order it appears to produce outputs. Instead, **it tries to build the code the reader will have**, and show the output of the code at that step of the tutorial.

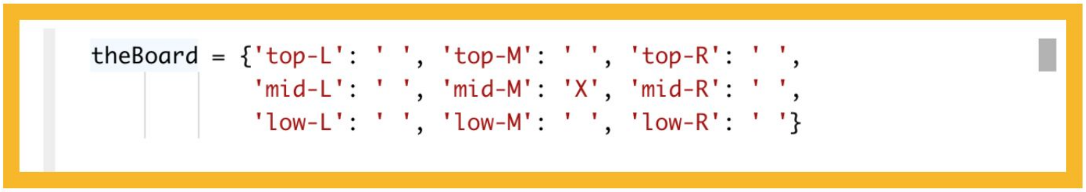
It does this by **collecting all the code that appears in a snippet** in the tutorial, and then **putting those snippets in the order they appeared in the source code file**.

For instance, try moving this cell…



```
print(theBoard)
```

Above this cell…



```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

This should break the output, because the "print" statement that uses theBoard will appear before theBoard is defined, right? *No!*

In fact, you should see that the output stays the same.

At each of the outputs in the tutorial, Torii adds the snippets together in the order they appeared *in the source code*. This means you can show code in whatever order you want, without needing to be constrained by showing it in the order it appeared in the original file.

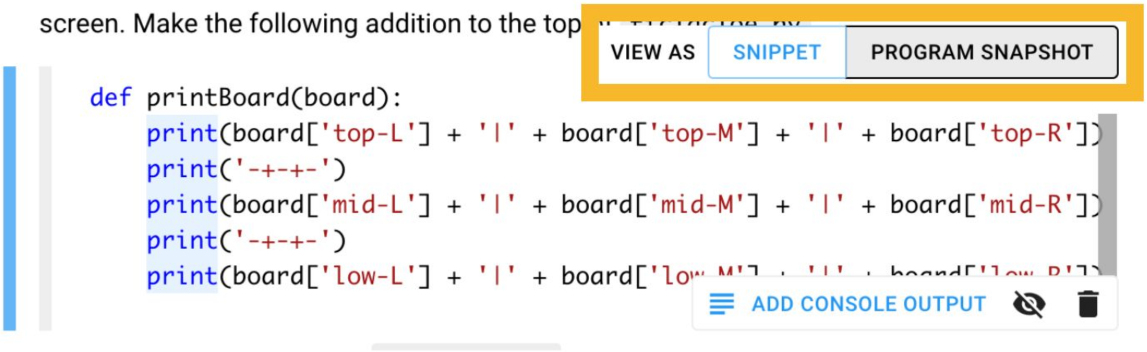
How can you tell what Torii is running to produce output? Open the "program snapshot" right before the output:

1. Find the first snippet above an output
2. Click it
3. Click on "View as Program Snapshot"

What do you see here? It's all of the code that's been shown in the tutorial so far, in the order it appeared in the original source file. This is what is run to produce the output right below it.

(continued on next page)

To see another example, look at the snapshot for this snippet.



screen. Make the following addition to the top

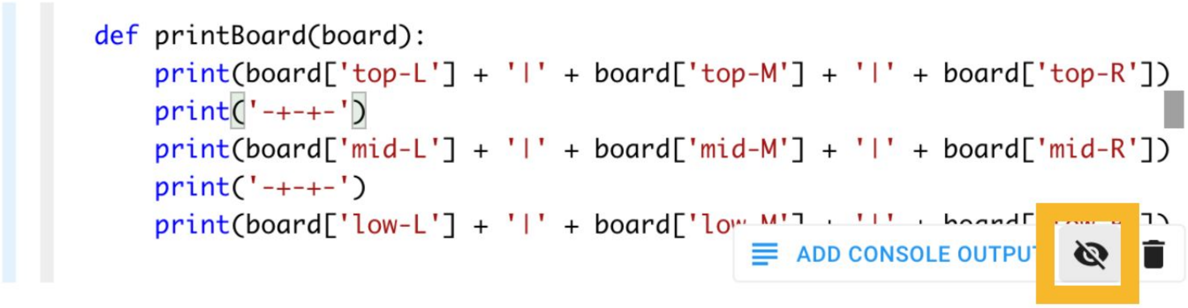
```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['lo
```

Notice how printBoard appears at the *top* of the snapshot, even though it was defined below the other code in the tutorial. This lets you, for instance, keep a source file where code has a very structured appearance, while giving you freedom to explain code in any order you want.

Step 6: Hide Distracting Code

Sometimes, you may want to hide code from the reader that might be distracting. You can hide code from the tutorial, while still using it to produce outputs, by clicking on a cell's "hide" button.

As an example, let's assume that you think users should already have printBoard in their file before they use your tutorial, and you want to hide it here, but keep it as part of the execution environment. Click the **hide** button for that cell.



```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['lo
```

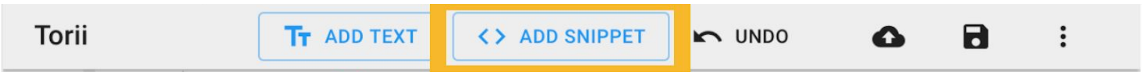
Step 7: Add Snippets, Add Outputs

In upcoming tasks, you will add new content to tutorials.

Let’s say you want to add a step that changes the board, and prints out the new board. Add these lines to the very very bottom of the “ticTacToe.py” file:

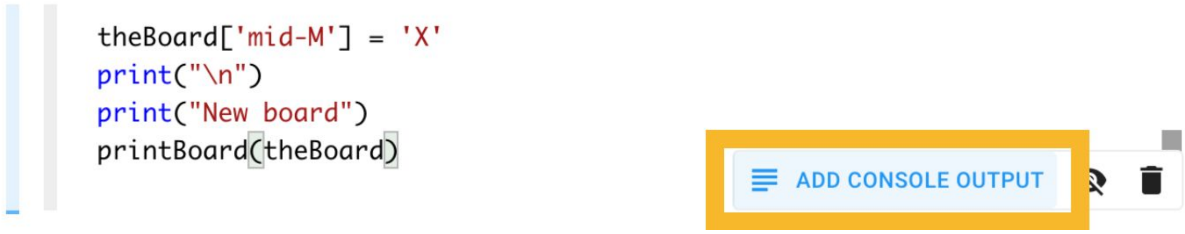
```
theBoard['mid-M'] = 'X'
print("\n")
print("New board")
printBoard(theBoard)
```

Select all four lines in the source code editor. Then click “**Add Snippet**”



A new snippet will appear in the tutorial, *below* the previously selected cell. If no cell was previously selected, it will appear at the top of the tutorial.

Then, **add an output** by clicking on **“Add Console Output”** on the new snippet.



```
theBoard['mid-M'] = 'X'
print("\n")
print("New board")
printBoard(theBoard)
```

The generated output is live. Try changing the code in the snippet and seeing the output change.

(continued on next page)

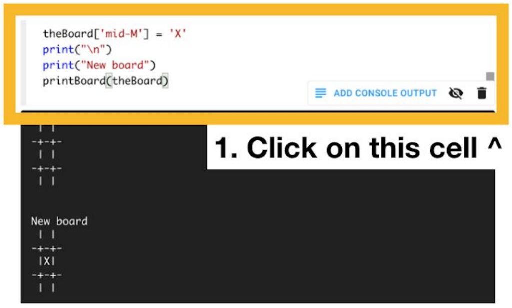
Step 8: Make Local Edits to Code

Sometimes, you will want to **change code** at some point in the tutorial that has appeared before. To change code that has appeared in the tutorial before, you must follow a few steps:

1. **Click the snippet** where you want to change previous code
2. Click “**View as Program Snapshot**”
3. Click the **place where you want to make a change**
4. Turn ***off*** the “**Sync Edits**” button

Now you can make changes to just that piece of code, which will show up in all future snippets.

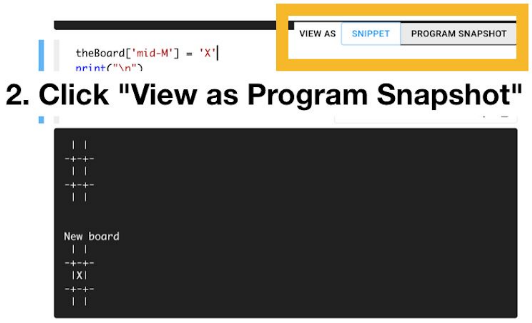
For example, let’s say you want to print out the text “Old board” before the first board is printed in the very last output. How might you do this?



```
theBoard['mid-M'] = 'X'
print("\n")
print("New board")
printBoard(theBoard)
```

```
-+-+-
 | |
-+-+-
 | |


New board
 | |
-+-+-
 |X|
-+-+-
 | |
```

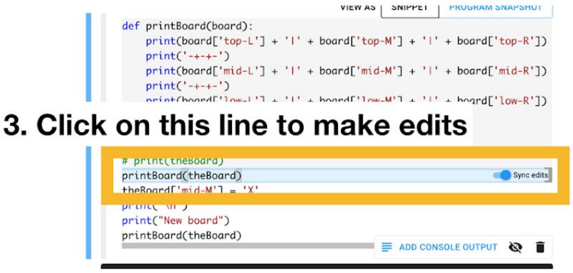
1. Click on this cell ^



```
theBoard['mid-M'] = 'X'
print("\n")
```

2. Click "View as Program Snapshot"

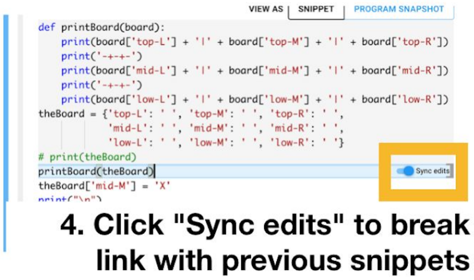
```
 | |
-+-+-
 | |
-+-+-
 | |


New board
 | |
-+-+-
 |X|
-+-+-
 | |
```



```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
```

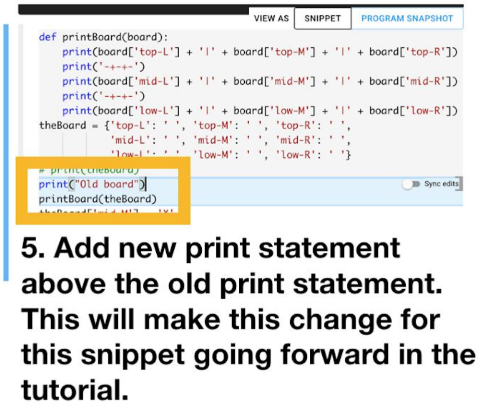
3. Click on this line to make edits

```
# print(theBoard)
printBoard(theBoard)
theBoard['mid-M'] = 'X'
print("\n")
print("New board")
printBoard(theBoard)
```



```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
# print(theBoard)
printBoard(theBoard)
theBoard['mid-M'] = 'X'
print("\n")
```

4. Click "Sync edits" to break link with previous snippets



```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
# print(theBoard)
print("Old board")
printBoard(theBoard)
```

5. Add new print statement above the old print statement. This will make this change for this snippet going forward in the tutorial.

That’s it! Have any questions? If so, ask the facilitator—we want to clear up as much about the tool and how it works with you as we can before the upcoming tasks.