SOFT40161_Lab01

October 1, 2025

Nottingham Trent University
Department of Computer Science

1 SOFT40161 - Introduction to Computer Programming

1.1 Introduction

1.2 Lab Learning Outcomes

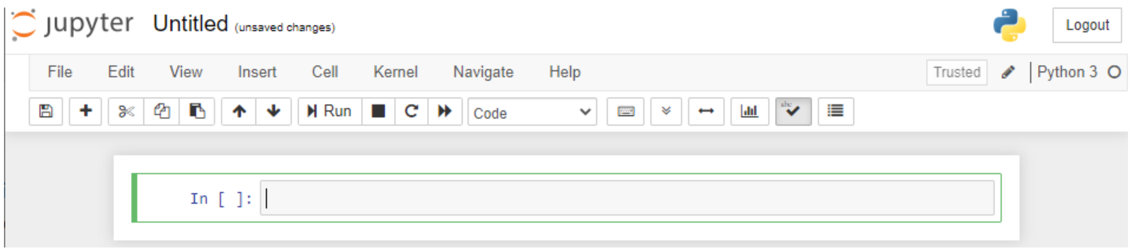
- Install Jupyter notebook correctly
- Use Jupyter notebook to view slides and write code.
- Know some common practices in developing Python programs.
- Write the first Python programs.

1.3 Introduction to Jupyter Notebook

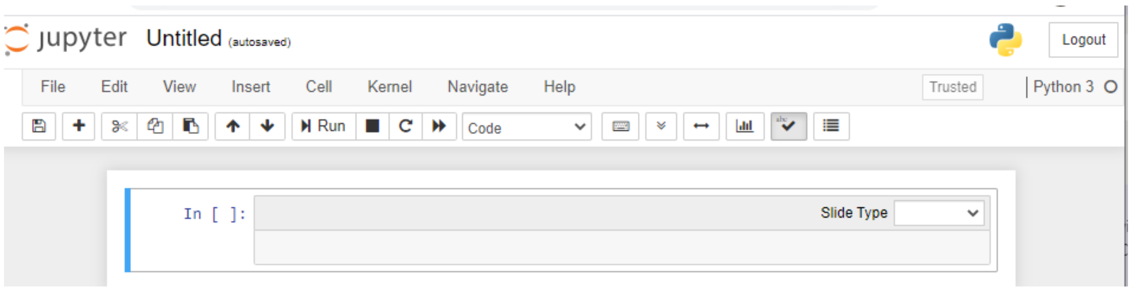
- Please note that information in this lab is mainly taken from
  - https://docs.jupyter.org/en/latest/
  - https://realpython.com/jupyter-notebook-introduction/
  - https://www.tutorialspoint.com/jupyter/jupyter_notebook_types_of_cells.htm
- The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.
- The name **Jupyter** comes from the core supported programming languages that it supports: **Ju**lia, **Pyt**hon, and **R** (of course, an **e** was added before **R**).
- Jupyter ships with the IPython kernel, which allows you to write your programs in Python.

1.3.1 Creating a Notebook

- After you start a Jupyter notebook, click on the New button (upper right), and it will open up a list of choices. Choose Python 3.
- A web page is created that may look like this:


jupyter Untitled (unsaved changes)
Logout
File Edit View Insert Cell Kernel Navigate Help
Trusted
Python 3
Run
Code
In [ ]:

- Now, go to **View** in the tool bar, click on **Cell Toolbar** and choose **Slide Show**. This will add a **Slide Type** at the upper right corner of each cell, which determines the type of slide it is. If you are interested in more information about this, please consult https://www.dev2qa.com/how-to-create-slideshow-from-jupyter-notebook/. You may need this only if you are planning to produce slides using Jupyter.


jupyter Untitled (autosaved)
Logout
File Edit View Insert Cell Kernel Navigate Help
Trusted
Python 3
Run
Code
In [ ]:
Slide Type

- Note that if you want to create more cells, you can press the + button at the upper left.

1.3.2 Naming a Notebook

- At the top of the page, there is the word **Untitled**. This is the title for the page and the name of your Notebook.
- To change the name of your notebook, move your mouse over the word **Untitled** and click on the text.
- You should now see an in-browser dialog titled Rename Notebook.
- Rename it to Hello Jupyter

1.4 Cell Modes in Jupyter Notebook

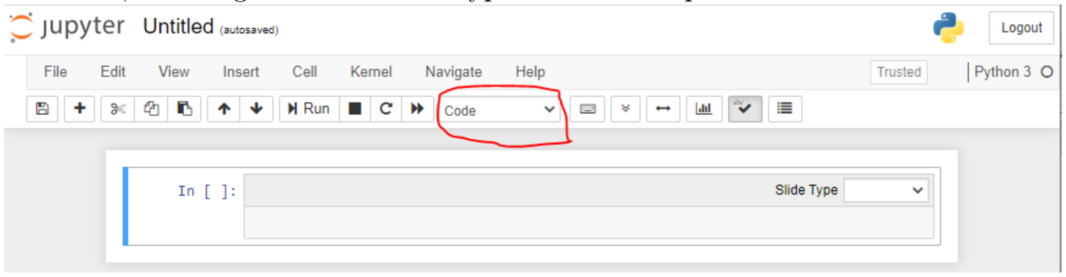
- Cells in Jupyter Notebook have two modes
  - **Edit** Mode
    - ∗ If the cell has a green border, that means you're in **Edit** mode.
    - ∗ Edit mode is for all the actions you would usually perform in the context of a single cell.
    - ∗ For example, editing and typing your code and text.
  - **Command** Mode
    - ∗ If the cell has a blue border, then you're in **Command** mode.
    - ∗ Command mode is for doing things outside the scope of any individual cell, often applying actions to multiple cells at once.
    - ∗ For example, you can select multiple cells, copy them, and paste them, or delete them if you like.
- One can get into **Edit** mode by pressing the < Enter > button or by double clicking on the cell. Pressing the **Esc**ape button will exit the **Edit** mode to become in **Command** mode.

2 Cell Types

- **Code** Cells
  - *Contents* in this cell are treated as statements in a programming language of the current kernel, in our case Python 3.
  - When such cell is run, its result is displayed in an output cell.
  - The output may be text, image, matplotlib plots or HTML tables.
- **Markdown** Cells
  - These cells contain text formatted using markdown language.
  - All kinds of formatting features are available like
    - making text bold (putting words between two **)
    - making text italic (putting words between two *)
    - displaying a heading (starting the line with a # followed by a blank)
    - displaying a subheading (starting the line with a ## followed by a blank)
    - displaying a subsubheading (starting the line with a ### followed by a blank)
    - displaying unordered list (starting the line with a hyphen - followed by a blank)
    - ... etc.
  - Markdown cells are especially useful to provide documentation to the computational process of the notebook.
  - For more information on Markdown, visit https://guides.github.com/features/mastering-markdown/
- **Raw** Cells
  - Contents in raw cells are not evaluated by notebook kernel, that is why it is better to use Markdown.

2.1 Changing Cell Types

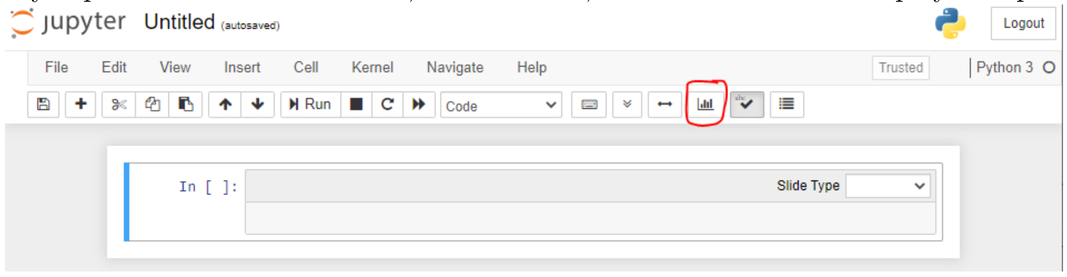
- Changing the type of a cell can be accomplished through the tool bar at the top of the notebook, choosing the desired cell type from the drop-down menu.



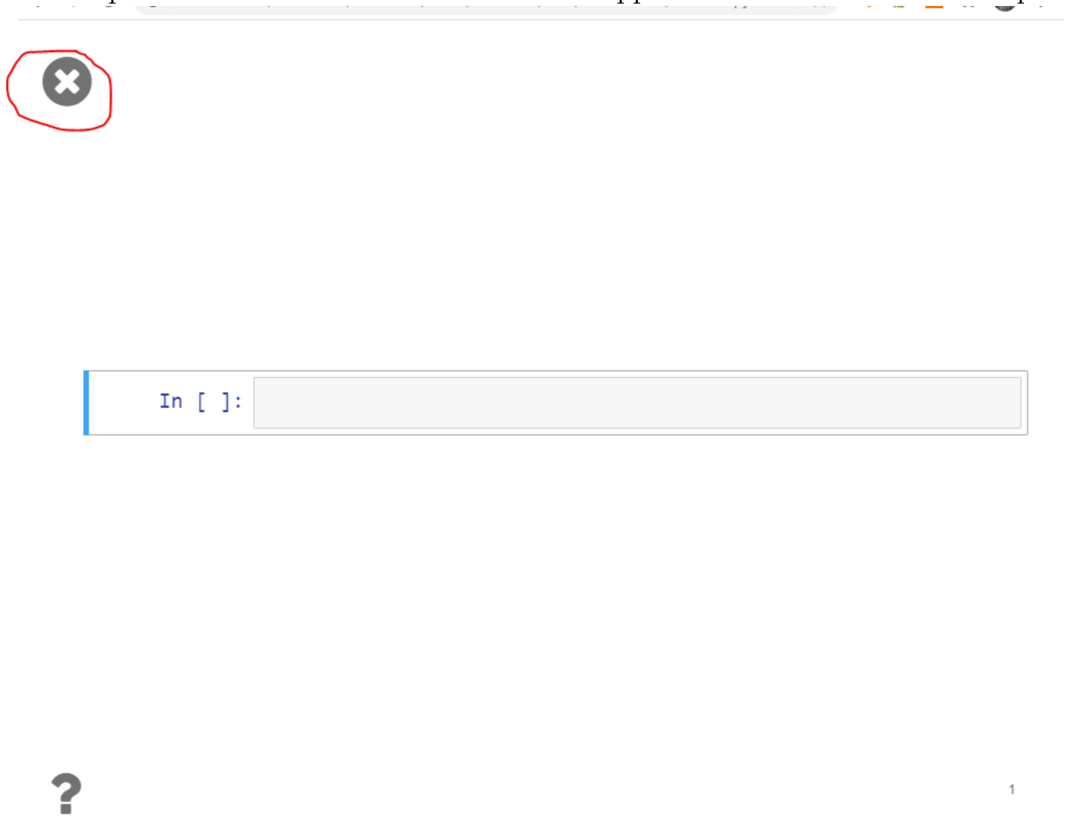
- Alternatively, while in a cell in **command** mode,
  - pressing the letter **y** will make the cell of type Code.
  - pressing the letter **m** will make the cell of type Markdown.
- Note that new cells are of type **Code**, by default, when they are created.

3 Displaying Jupyter Notebook as a Presentation

- If you press the bar chart button, circled below, the notebook will be displayed as presentation.


jupyter Untitled (autosaved)
Logout
File Edit View Insert Cell Kernel Navigate Help
Trusted
Python 3
Run
Code
In [ ]:
Slide Type

- If you want to go outside the presentation mode, you can press the **X** at the upper left of the presentation.



3.1 Handling Code Cells

- In order to run a cell containing code, press < Shift >+< Enter > buttons simultaneously.
- In order to clear the output, go to **Cell** at the tool bar, then choose **Current Outputs**, then choose **Clear**.
- In order to save the contents of a cell in a file called **filename.py**,
  - add the following line as the first line in the cell (before all your code) %%writefile filename.py
  - Now, if you press < Shift >+< Enter > buttons simultaneously, it will save the contents of the cell, instead of compiling and running the code in the cell.

- For example, go ahead and save the contents of the following cell in a file called `myFirstProgram.py`.
  – Note that if you want to run the code, you can simply comment the first line (or remove it).

[18]:
```
#%%writefile myFirstProgram.py
a = 1
b = 3
print(a+b)
```

```
4
```

4 Some Common Practices and Notes

When writing python statements, one can - write each statement in a single line as follows

[20]:
```
 # Statements at different lines
a = 1
b = 2
a + b
```

[20]:
```
3
```

- write multiple statements separated by semi colon(s) in one line as follows

[22]:
```
# Statements on the same line
a = 10; b = 20; a * b
```

[22]:
```
200
```

- use another cell for computing the results (make sure you note the last cell that was executed).
- It is good to execute a far away cell and then execute the last cell, to show that the order to execution is the most important factor, not the physical location in the jupyter notebook.

[25]:
```
a = 12
b = 3
a+b
```

[25]:
```
15
```

[26]:
```
a - b
```

[26]:
```
9
```

- determine the type of a variable (we will know about variables later)

[28]:
```
a='7.8'
type(a)
```

```
a=5.6
a
```

```
[28]: 5.6
```

```
[29]: c = a // b
      print(c)
      type (c)
```

```
1.0
```

```
[29]: float
```

```
[30]: print(type(12))
      print(type(3))
      print(15.7/4.4)
      print(15.7//4.4)
      print(12.5/3)
```

```
<class 'int'>
<class 'int'>
3.5681818181818183
3.0
4.166666666666667
```

5 Task 1: Playing with Markdown

- Copy the contents of the text file `MarkDownExample.txt` into a Markdown cell. Note the different colors for different types of lines.
- Press the < Shift >+< Enter > buttons simultaneously, you should now see it rendered, based on the line type.
- Double click on the cell to go back to Markdown text.

6 Cell Types

- Code Cells
  - Contents in this cell are treated as statements in a programming language of the current kernel, in our case Python 3.
  - When such cell is run, its result is displayed in an output cell.
  - The output may be text, image, matplotlib plots or HTML tables.
- Markdown Cells
  - These cells contain text formatted using markdown language.
  - All kinds of formatting features are available like
    * making text bold (putting words between two **)
    * making text italic (putting words between two *)
    * displaying a heading (starting the line with a #)
    * displaying a subheading (starting the line with a ##)

        * displaying a subsubheading (starting the line with a ###)
        * displaying unordered list (starting the line with a hyphen -)
        * ... etc.
    - Markdown cells are especially useful to provide documentation to the computational process of the notebook.
    - For more information on Markdown, visit https://guides.github.com/features/mastering-markdown/
- Raw Cells
    - Contents in raw cells are not evaluated by notebook kernel, that is why it is better to use Markdown.

```
[ ]:
```

7 Task 2: From Celsius to Fahrenheit

Assign a value to a temperature in Celsius, and find and display its equivalent temperature in Fahrenheit according to the formula:

$$Fahrenheit = 32 + \frac{9}{5} * Celsius$$

```
[35]: #%%writefile Lab01Task2.py
      ## Uncomment the above line after you finish your code and want to save it in a␣
       ↪file.

      # Setting the Celsius value first
      Celsius = 100

      # Write your code below this line
```

```
[ ]:
```

8 Task 3: Finding the Area and Circumference of a Circle

Assign a value to a radius of a circle, then find and display its area and circumference using the following formulas

$$Area = pi * radius * radius$$

$$circumference = 2 * pi * radius$$

where

$$pi = 3.14159$$

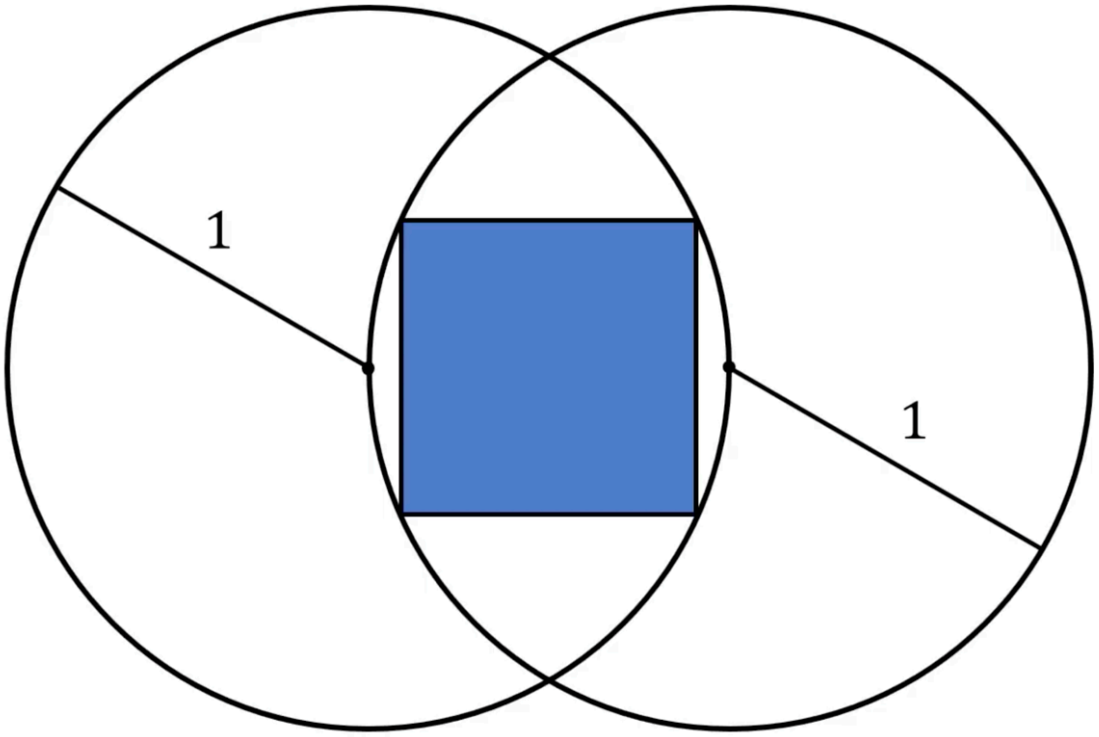
```
[ ]: #%%writefile Lab01Task3.py
     ## Uncomment the above line after you finish your code and want to save it in a␣
      ↪file.

     # Defining the constant PI
```

```
PI = 3.14159

# Setting the value of the radius
radius = 10.3

# Write your code below this line
```

9 Task 4: Finding the Area of the shaded region.

Two identical circles of radius 1 are drawn so that each circle passes through the centre of the other. A square is inscribed inside the overlapping region (lens-shaped area), with its vertical sides touching both circles. Can you calculate the shaded area? If we change the radius 1 to r, will you be able to write a code to calculate the shaded area for any given value of r? where

$$pi = 3.14159$$



[27]:
```
#%%writefile Lab01Task4.py
## Uncomment the above line after you finish your code and want to save it in a␣
↪file.

# Setting the value of the radius
# Write your code below this line
```

[ ]:

```

```