

A Gentle Introduction to Python

5c22 Computational Methods

Small history of Python

- Python is released by Guido Van Rossum (Jan 1989)
 - V1 release (1994)
 - V2 release (2000)
 - V3 release (2008) [*This is what we will be using!!*]
- Python is named after the hit comedy series Monty Python ([link](#))
- Open source from day 1
- Due to its open codebase and its “relatively easy” code structure and levels of abstraction
 - Very popular for creating fast working prototypes
 - Loads of open source libraries for almost all domains of expertise
 - The go to tool in the “modern” era for data processing

What is Python?

- Python is a high level, general purpose development language
 - Great deal of libraries that are constantly updated to help with whatever task you have
 - Tensorflow / Pytorch - Deep Learning
 - Matplotlib / Seaborn - Data Visualization
 - Plotly - Dashboards
 - Scipy / Numpy - Matrix operations
 - PyGame - Making python based video games!
- Python is designed with emphasis on code readability
 - Very good for getting things working quickly, and allowing others to understand what you've done
- Python is an **interpreted** language, meaning that it is **much slower** than compiled languages (C++), so there are certain use-cases where Python will not be ideal

How can we use Python?

Recommended (Basic) Setup:

1) Python Interpreter

- The lab machines *should* already have Python installed.
- If you want to install Python on your own machine, follow the steps outlined: <https://www.python.org/downloads/>

2) Code editor

- Visual studio code (VS Code) is a popular editor with loads of extensions available
- VS Code *should* already be installed on your lab machines
- Link to install on your own machine: <https://code.visualstudio.com/>

3) Python Extension for VS Code (optional but recommended)

- Visit the Extension Marketplace in VS Code
- Search for Python and install the “*Python*” extension (verified by Microsoft)

Setting up a workspace

- Create a new folder somewhere in your machine to house all the files you need for this project
 - I recommend creating a “*PythonTutorial*” folder in your “*Documents*” folder (“*Documents/PythonTutorial*”)
- Open this folder using VS Code
 - File -> Open Folder -> Navigate to “*PythonTutorial*” and open

Running your first program

- Create a new file in VS Code and name it “main.py”
 - Right-click in the File Explorer in VSCode and choose new file
- Remember the “*Hello World*” Demo from your first day of programming?
 - It’s even easier in Python!
 - Print any text you want on the screen by using the print() command



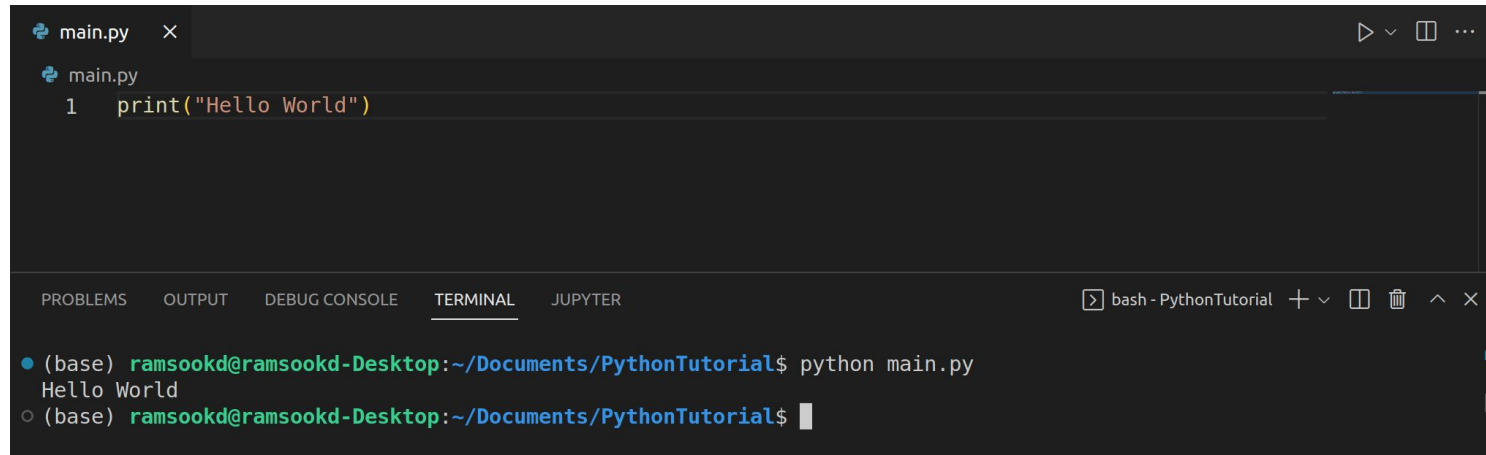
The screenshot shows a VS Code editor window with a dark theme. At the top, a tab is labeled 'main.py' with a Python icon on the left and a close button (X) on the right. Below the tab, the editor area shows the file 'main.py' with a single line of code: '1 print("Hello World")'. The line number '1' is on the left, and the code is on the right. The text 'print' is in blue, 'Hello World' is in red, and the parentheses and quotes are in yellow.

main.py

```
1 print("Hello World")
```

Running your first program - 2

- Now that we have a working program (“main.py”), we can run this through the Python Interpreter
- In VS Code, open a terminal window
 - In the toolbar click on Terminal -> New Terminal
- Execute “main.py” by running: `python main.py`



```
main.py x
main.py
1 print("Hello World")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
bash - PythonTutorial + v [] [] ^ x

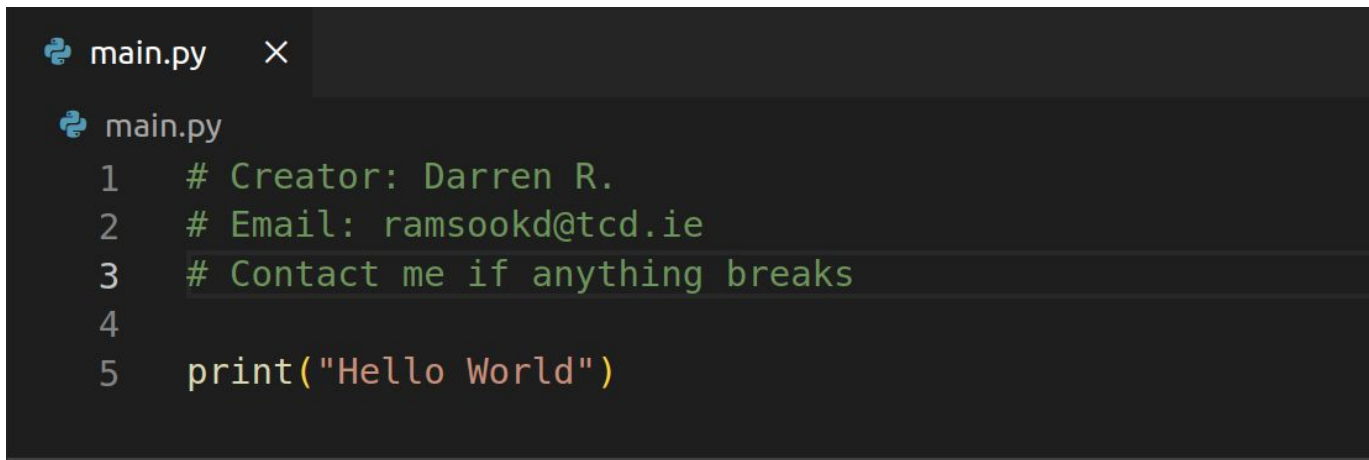
● (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py
Hello World
○ (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$
```

Congratulations!

- Congratulations on your first Python Program
 - You can now put “Python Programmer” on your CV
- But wait, all we did was use the `print()` function
 - There has to be more? Right?

Comments

- Adding comments to code can help you (*and others!*) understand the general workflow of your program
 - Add comments by using the # symbol *WRITE WHAT CODE IS SUPPOSED TO DO*

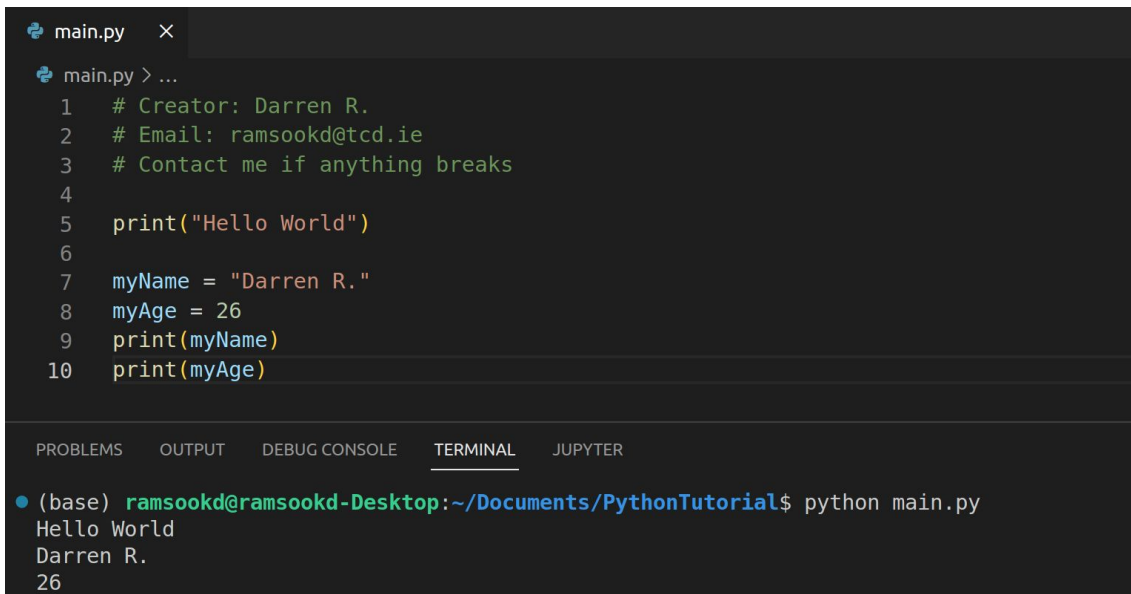


The screenshot shows a code editor window with a dark background. At the top, there is a tab labeled 'main.py' with a Python icon and a close button. Below the tab, the code is displayed with line numbers 1 through 5. Lines 1, 2, and 3 contain comments starting with '#'. Line 4 is empty. Line 5 contains a print statement. The text is color-coded: comments are green, and the print statement is orange and black.

```
main.py ×  
  
main.py  
1  # Creator: Darren R.  
2  # Email: ramsookd@tcd.ie  
3  # Contact me if anything breaks  
4  
5  print("Hello World")
```

Variables

- Variables are containers used for storing data values
 - Python has no command for declaring variables, instead they are created the moment you assign a value to it
 - Use variables by: *variableName = variableValue*



The image shows a Jupyter Notebook interface with a dark theme. The top part is a code editor for a file named 'main.py'. It contains a Python script with comments and code to print 'Hello World', a name, and an age. The bottom part is a terminal window showing the command to run the script and its output.

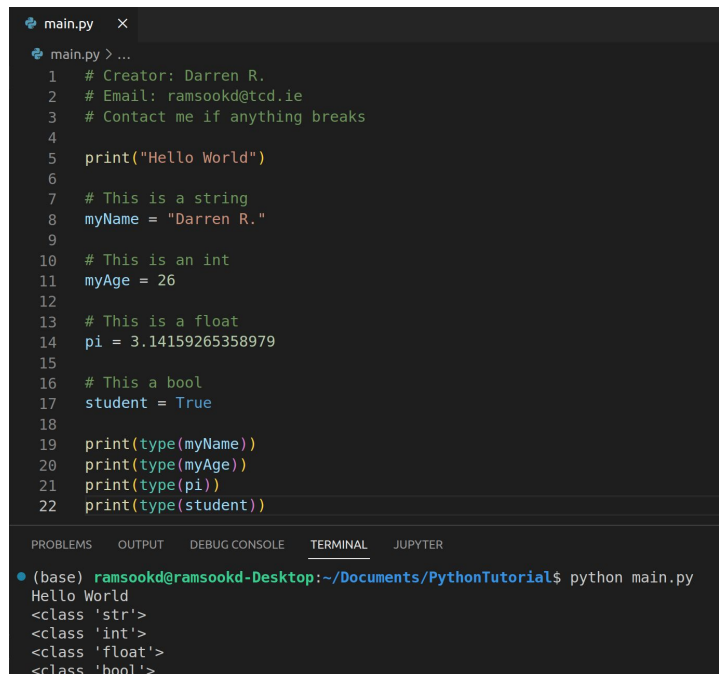
```
main.py x
main.py > ...
1  # Creator: Darren R.
2  # Email: ramsookd@tcd.ie
3  # Contact me if anything breaks
4
5  print("Hello World")
6
7  myName = "Darren R."
8  myAge = 26
9  print(myName)
10 print(myAge)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
● (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py
Hello World
Darren R.
26
```

Data Types

- Python is stocked full with multiple different datatypes
 - Python will understand the data type you intend to use via your variable declaration
- Some of the basic data types are:
 - *string* - Used for storing text characters
 - *int* - Used for storing Integer values
 - *float* - Used to store 64 bit numbers
 - *bool* - True or False, binary type data



The image shows a code editor window with a file named `main.py`. The script contains several comments and variable declarations for different data types, followed by `print` statements to verify their types. The terminal output shows the execution of the script, printing 'Hello World' and the type of each variable.

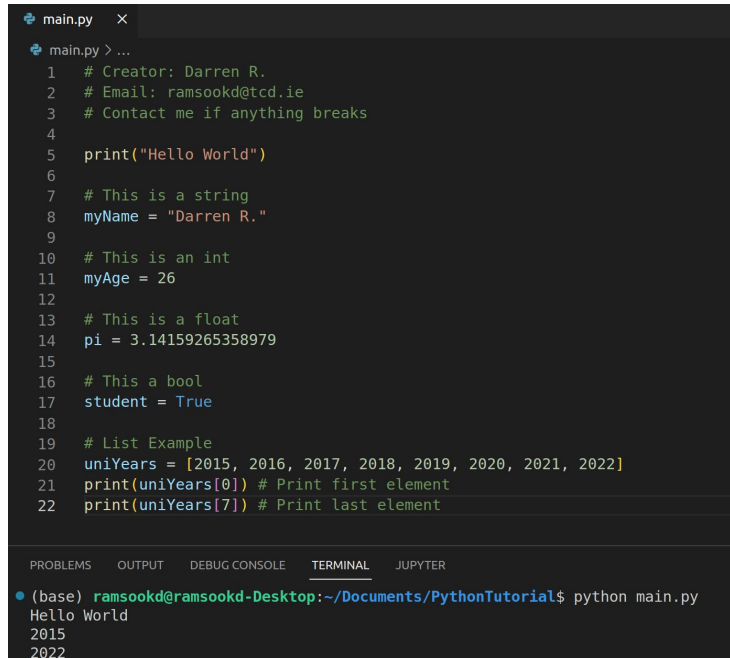
```
main.py X
main.py >...
1 # Creator: Darren R.
2 # Email: ramsookd@tcd.ie
3 # Contact me if anything breaks
4
5 print("Hello World")
6
7 # This is a string
8 myName = "Darren R."
9
10 # This is an int
11 myAge = 26
12
13 # This is a float
14 pi = 3.14159265358979
15
16 # This a bool
17 student = True
18
19 print(type(myName))
20 print(type(myAge))
21 print(type(pi))
22 print(type(student))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
• (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py
Hello World
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Data Structures in Python - List

- Lists are used in python to store a sequence of data elements
 - Lists are created using: `listName = ['a', 'b', 'c', 'd']`
 - Lists can be indexed by: `listName[x]`, where x is the index of the list
 - Lists indexing starts at 0 and goes to $N-1$, where N is the length of the list

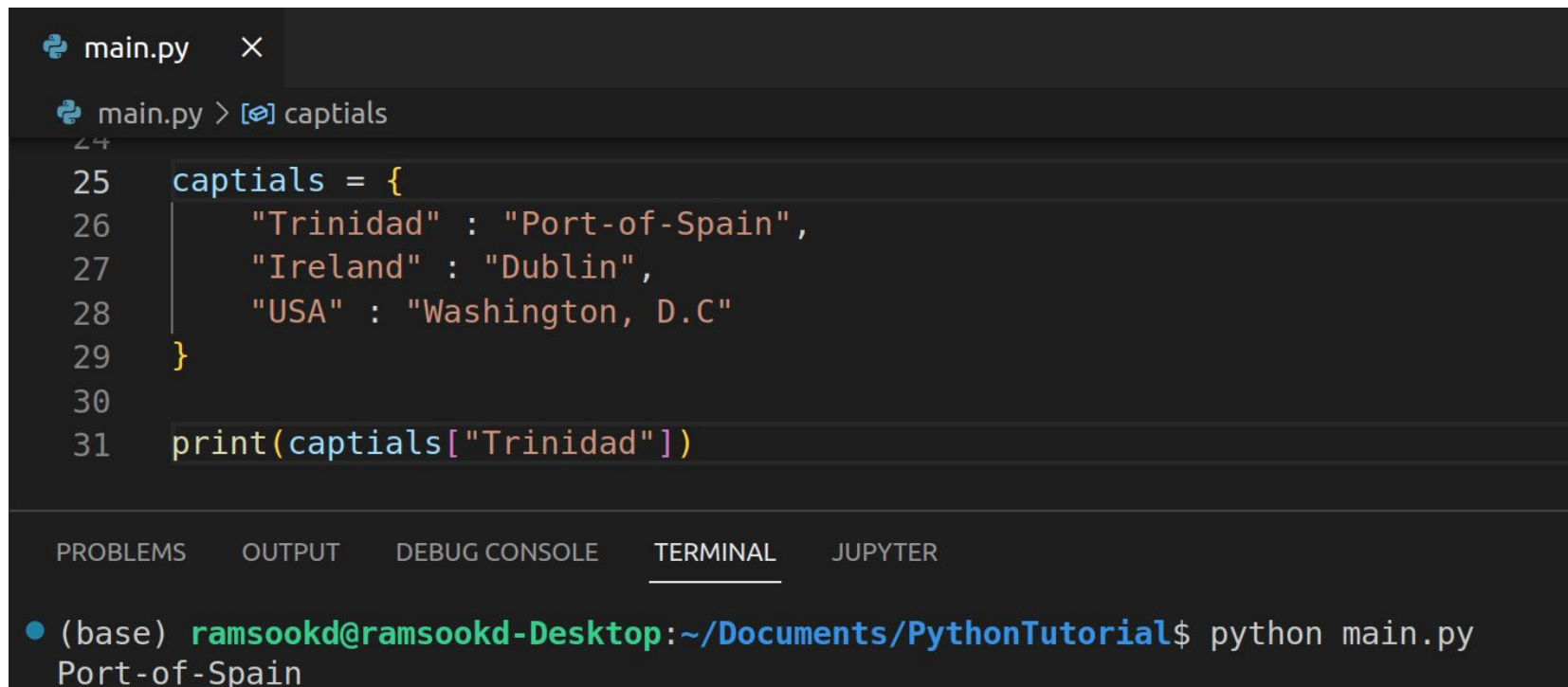


```
main.py x
main.py > ...
1 # Creator: Darren R.
2 # Email: ramsookd@tcd.ie
3 # Contact me if anything breaks
4
5 print("Hello World")
6
7 # This is a string
8 myName = "Darren R."
9
10 # This is an int
11 myAge = 26
12
13 # This is a float
14 pi = 3.14159265358979
15
16 # This a bool
17 student = True
18
19 # List Example
20 uniYears = [2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]
21 print(uniYears[0]) # Print first element
22 print(uniYears[7]) # Print last element

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
• (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py
Hello World
2015
2022
```

Data Structures in Python - Dictionary

- Dictionaries are used to store data in key:value pairs
 - Note: Keys must be unique!



The screenshot shows a code editor with a file named `main.py`. The code defines a dictionary `captials` with three key-value pairs: "Trinidad" : "Port-of-Spain", "Ireland" : "Dublin", and "USA" : "Washington, D.C". It then prints the value for the key "Trinidad". Below the code editor, the terminal output shows the command `python main.py` being executed, resulting in the output `Port-of-Spain`.

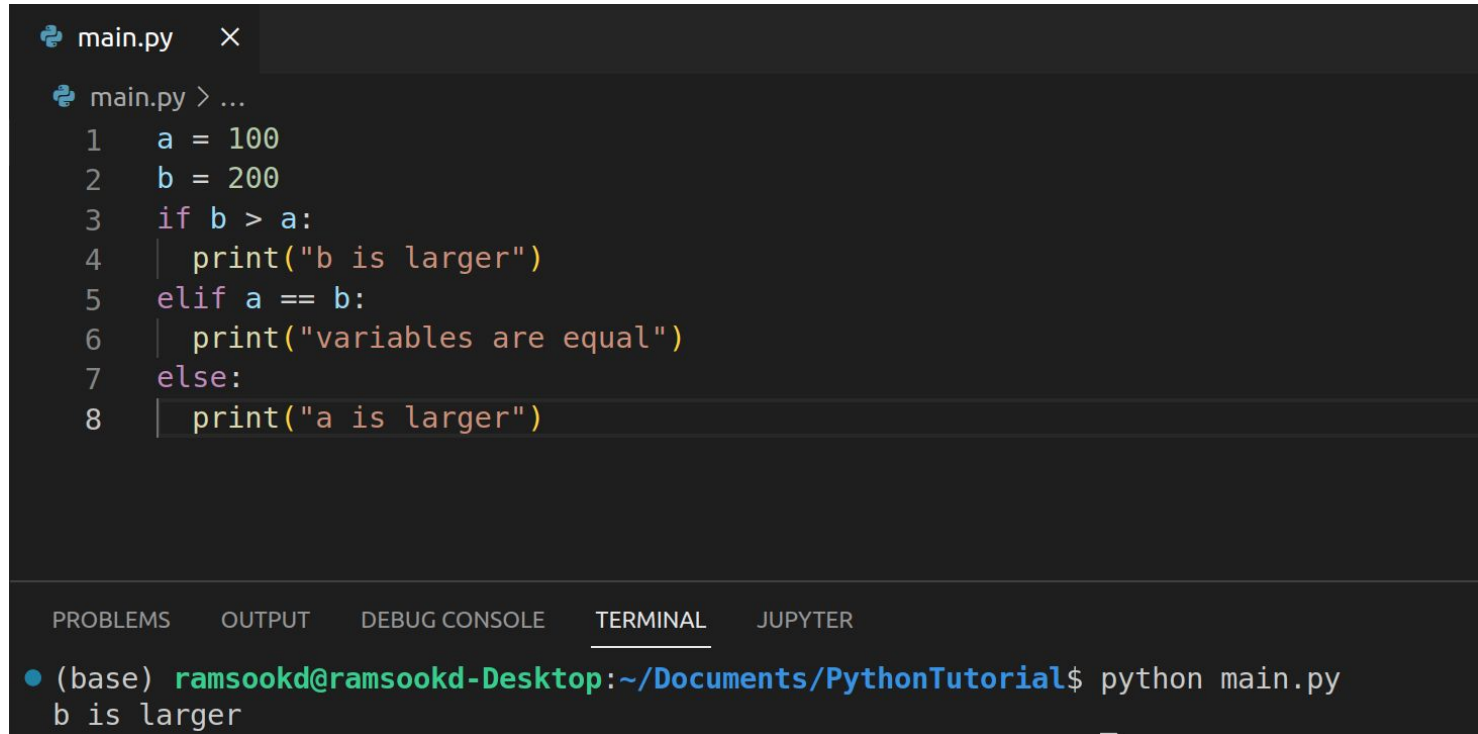
```
main.py ×  
main.py > [?] captials  
25 captials = {  
26     "Trinidad" : "Port-of-Spain",  
27     "Ireland" : "Dublin",  
28     "USA" : "Washington, D.C"  
29 }  
30  
31 print(captials["Trinidad"])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
● (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py  
Port-of-Spain
```

Control Sequences

- If ... else

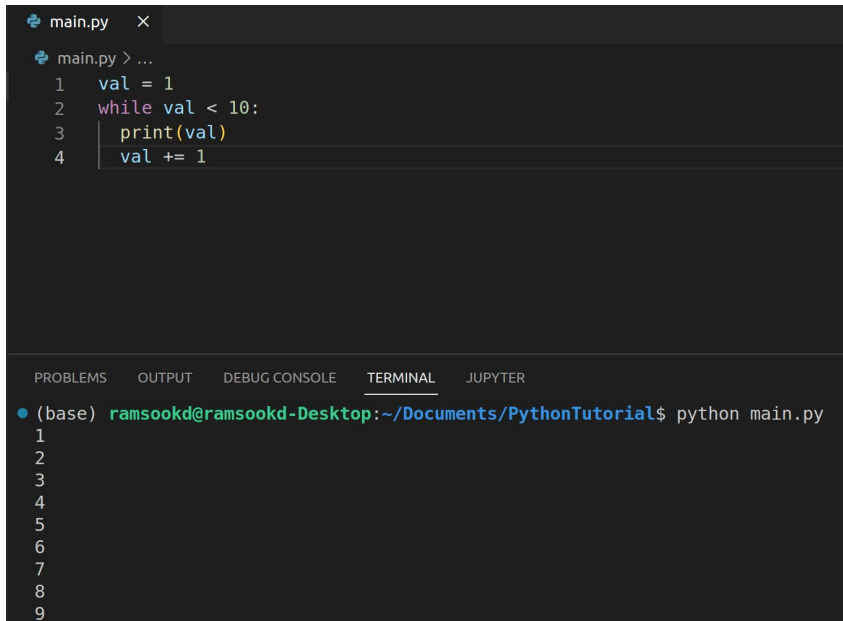


The screenshot shows a code editor with a file named `main.py`. The code defines two variables, `a` and `b`, and uses an `if-elif-else` statement to compare them. The output of the script is displayed in the terminal panel below the code editor.

```
main.py  ×  
main.py > ...  
1  a = 100  
2  b = 200  
3  if b > a:  
4      print("b is larger")  
5  elif a == b:  
6      print("variables are equal")  
7  else:  
8      print("a is larger")  
  
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER  
• (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py  
b is larger
```

Control Sequences - While Loops

- Execute some code while a condition is True
 - Getting stuck in while loops can be very dangerous!
 - Use Ctrl+C to terminate program if this happens



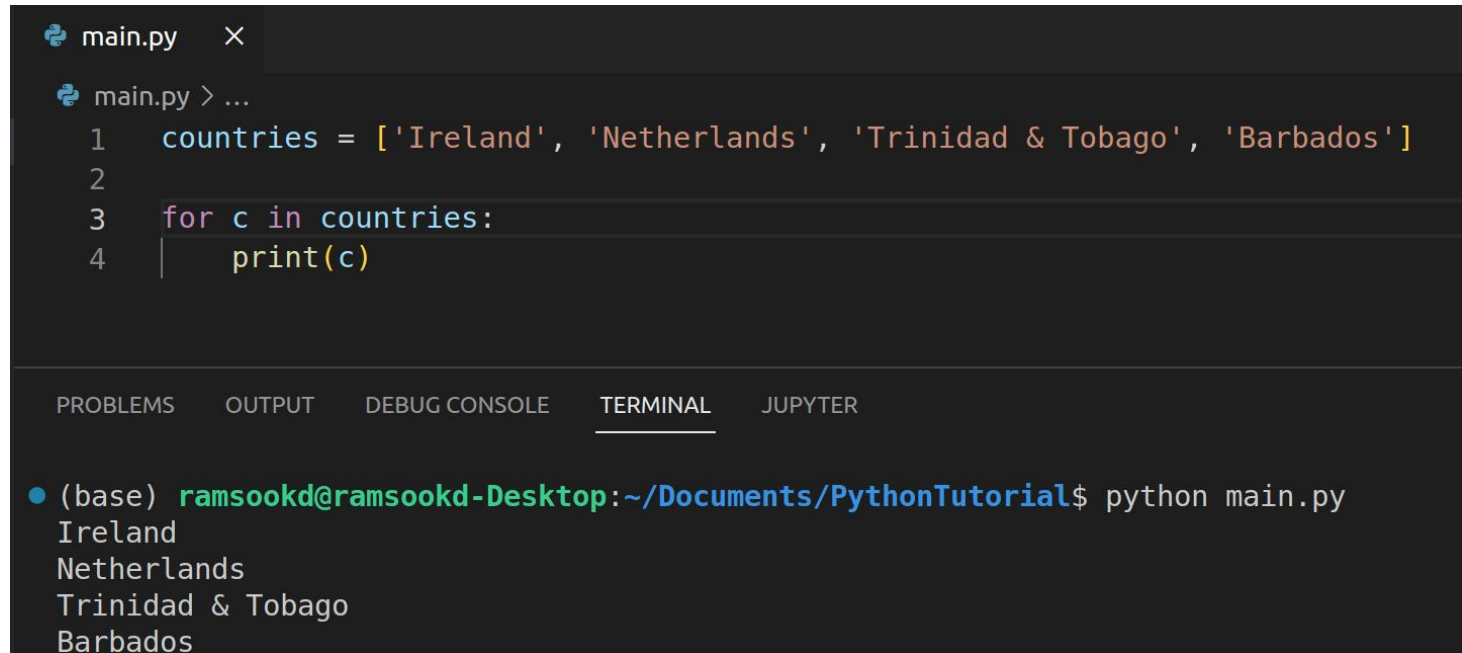
The image shows a code editor window with a file named `main.py`. The code in the editor is as follows:

```
1 val = 1
2 while val < 10:
3     print(val)
4     val += 1
```

Below the code editor is a terminal window. The terminal shows the command `python main.py` being executed, and the output is the numbers 1 through 9, each on a new line. The terminal prompt is `(base) ramsookd@ramsookd-Desktop: ~/Documents/PythonTutorial$`.

Control Sequences - For Loops

- Use a for loop for iterating over a sequence

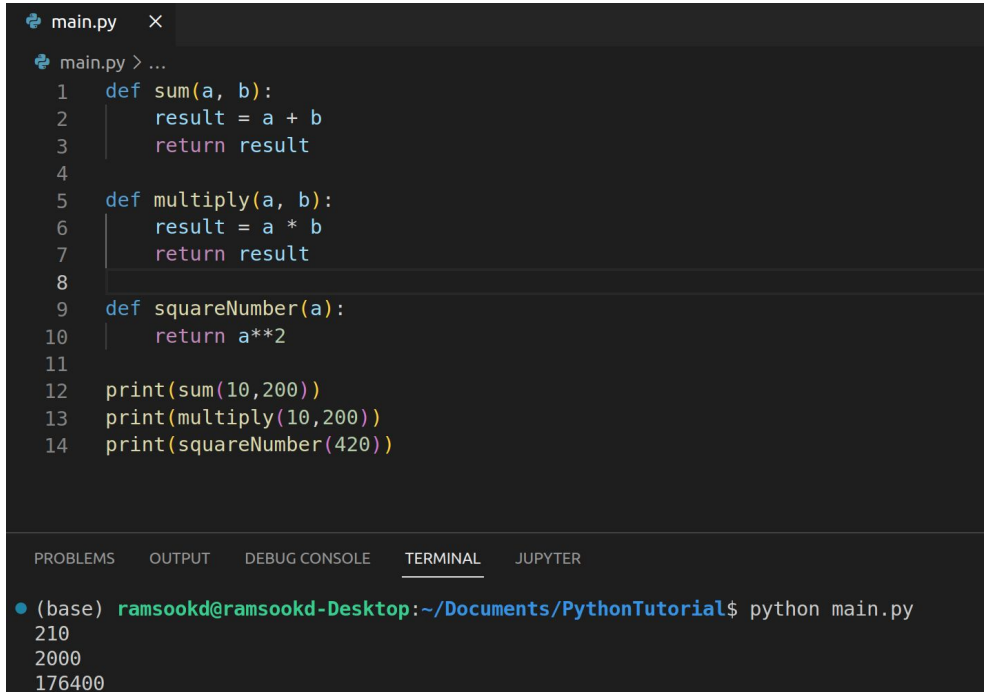


The image shows a Jupyter Notebook interface with a dark theme. At the top, there's a tab labeled 'main.py' with a close button. Below the tab, the code editor shows a Python script with four lines: line 1 defines a list 'countries' with four country names; line 2 is empty; line 3 starts a 'for' loop 'for c in countries:'; and line 4 has an indented 'print(c)' statement. Below the code editor is a horizontal bar with five tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'JUPYTER'. The 'TERMINAL' tab shows the command prompt output: a prompt followed by the command 'python main.py' and the four country names printed on separate lines.

```
main.py ×  
main.py > ...  
1 countries = ['Ireland', 'Netherlands', 'Trinidad & Tobago', 'Barbados']  
2  
3 for c in countries:  
4     print(c)  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER  
  
• (base) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py  
Ireland  
Netherlands  
Trinidad & Tobago  
Barbados
```


Creating Functions

- Functions are useful for creating code blocks that will be used in multiple scenarios

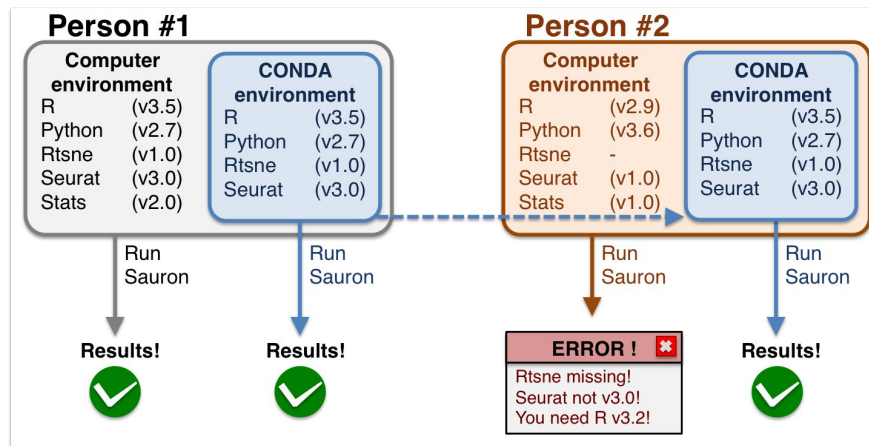


```
main.py x
main.py > ...
1 def sum(a, b):
2     result = a + b
3     return result
4
5 def multiply(a, b):
6     result = a * b
7     return result
8
9 def squareNumber(a):
10    return a**2
11
12 print(sum(10,200))
13 print(multiply(10,200))
14 print(squareNumber(420))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
• (base) ramsookd@ramsookd-Desktop: ~/Documents/PythonTutorial$ python main.py
210
2000
176400
```

Revisiting our setup

- As stated before, Python's real power comes from its open community the libraries available
- However things can get messy if we just start installing libraries randomly
- To help “containerize” this, we can use conda to help separate our projects



Conda

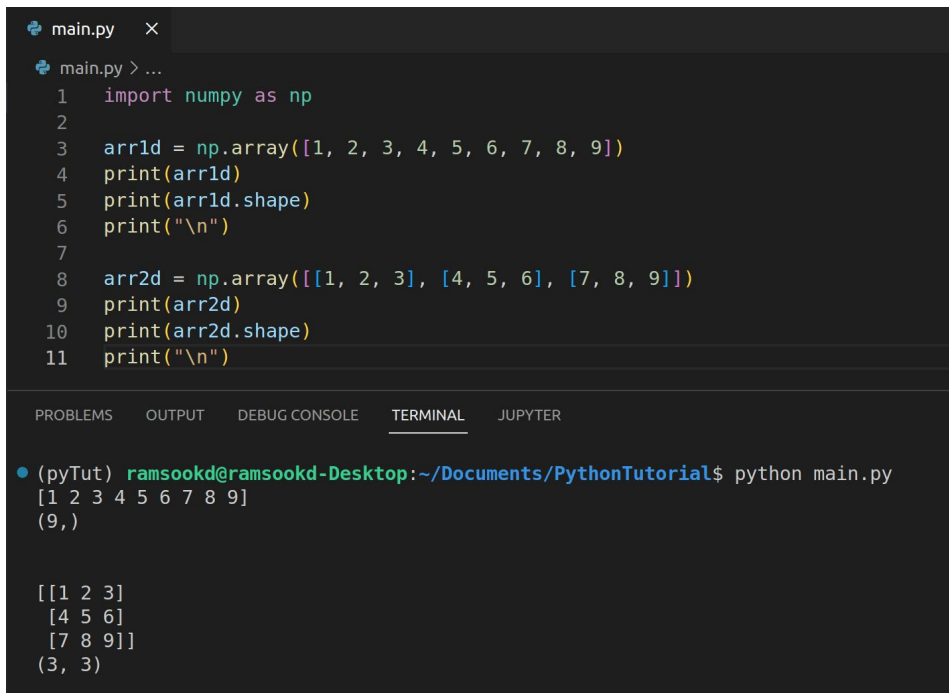
- Conda *should be* installed on your lab machines already
 - If not, you can install conda on your machine:
<https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html>
- I also have this on standby whenever I use conda:
https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf
 - It's a conda cheat sheet!

Creating an environment for our tutorial

- In our next exercise, we want to use numpy
 - However we don't want to install numpy for all Python projects, but just for our tutorial exercise
- We can solve this by using conda
 - First create a conda virtual environment: *conda create --name pyTut python=3*
 - Then activate this virtual environment by using: *conda activate pyTut*
- Once we have the virtual environment “pyTut” activated, all installed packages/libraries will be only installed to pyTut
 - Lets install numpy by running: *pip install numpy*

Using Numpy

- Numpy is a library created with the use of array mathematics in mind
 - Uses C/C++ backend for doing array calculations



The screenshot shows a Jupyter Notebook interface with a dark theme. The top part displays the code for creating two Numpy arrays: a 1D array 'arr1d' and a 2D array 'arr2d'. The bottom part shows the output of running the code, which prints the arrays and their shapes.

```
main.py x
main.py > ...
1 import numpy as np
2
3 arr1d = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
4 print(arr1d)
5 print(arr1d.shape)
6 print("\n")
7
8 arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
9 print(arr2d)
10 print(arr2d.shape)
11 print("\n")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
• (pyTut) ramsookd@ramsookd-Desktop:~/Documents/PythonTutorial$ python main.py
[1 2 3 4 5 6 7 8 9]
(9,)
```



```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
(3, 3)
```

We've finished our development

- But how can we be sure someone else can run our code?
 - We installed NumPy, but not everyone has NumPy!
 - In a real project, you may have multiple different libraries
 - Ensuring that someone else is able to run your code is crucial
 - A list of installed packages can be generated by `pipreqs`!
-
- Install `pipreqs` through: *`pip install pipreqs`*
 - Now you generate a list of installed packages via:
 - *`pipreqs .`*
-
- This will generate a `requirements.txt` file in your working directory