Contents

1 Welcome to Python!

Python is a widely used programming language, used in many fields, including data science, machine learning, and even math animations (3Blue1Brown's animations are done in Python!). A programming language is simply a medium through which we communicate with a machine - in this case, our computer. Many concepts in Python are common and found in other programming languages - so getting familiar with a concept in Python will help you get acquainted with other programming languages in the future.

A Python program consists of a sequence of statements, executed in turn, much like how you may read a book. When we run and execute Python code, the computer reads the code top to bottom, and executes it line by line. Throughout the rest of the labs, you will see code cells like the one below, which are written in Python. Every code block in this lesson can be found in a separate code file, so that you can run, edit, and experiment with the code.

```python
module = "PS0001"
lab = 1
author = "Caleb Tay"
print("Hello, and welcome to", module, "week", lab)
print("This lab was written and developed by", author)
```

When run, the code above, as the words suggest, *prints* (displays) to you, the user, some text. You will be learning how to write Python code like the above throughout this course.l

1.1 Python and `.py` files

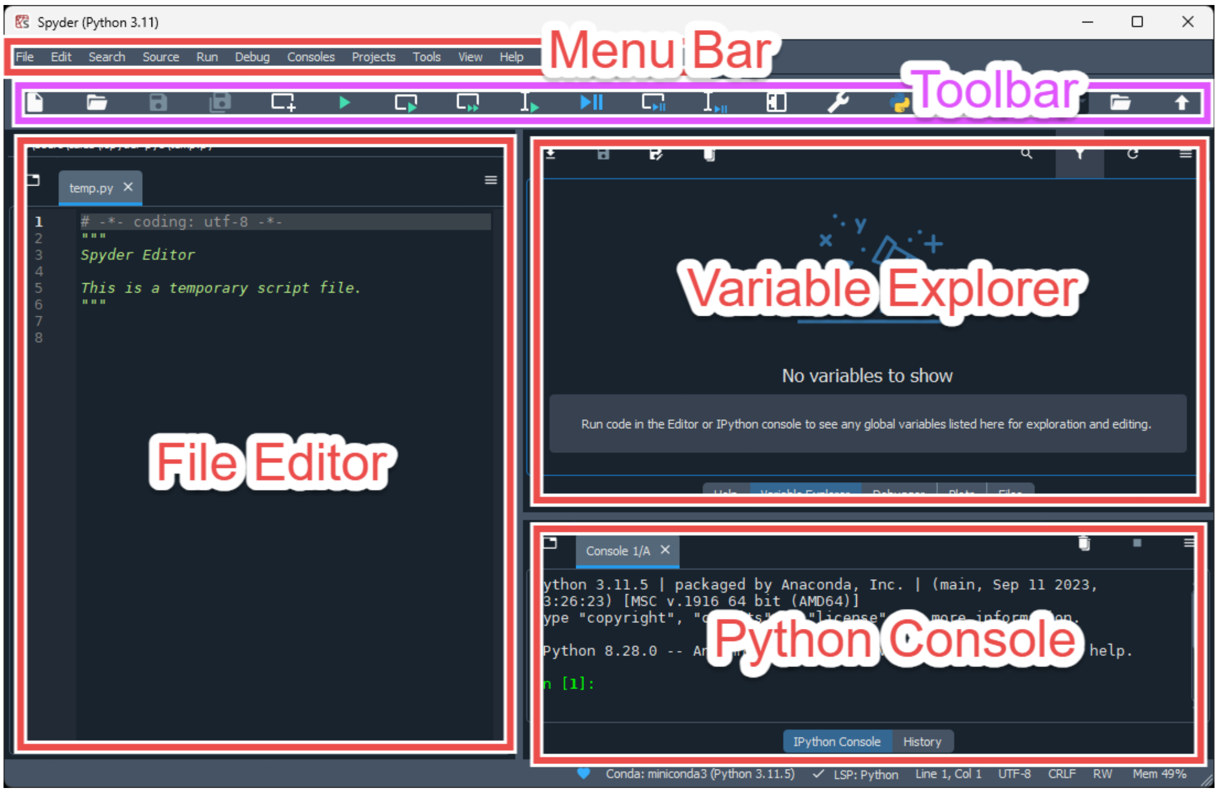
Before we move on, let me clarify sthe difference between these three. Python is the **programming language** that we are using for this course. It has its own syntax that must be followed strictly. Over the next 6 weeks, we will focus on learning Python syntax. We interact with it through different interfaces, including Spyder and the console. In other contexts, you may also use Jupyter Notebooks to interact with Python.

`.py` files are Python *scripts*, written with Python and following Python's syntax. You can open and edit these files in a normal text editor (e.g. Notepad on Windows, TextEdit on macOS), or in an application like Spyder. You can even rename text files (for example, `names.txt`) to have a `.py` extension (`names.py`). It is still a file that contains text, just associated with a different program to open or run with.

2 Spyder's user interface

Make sure that you are on Spyder's default window layout. Use the menu bar to navigate to View > Window layouts > Default layout.

There are **five** important elements in this interface, highlighted and labelled in Figure 1.



```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
```

No variables to show

Run code in the Editor or IPython console to see any global variables listed here for exploration and editing.

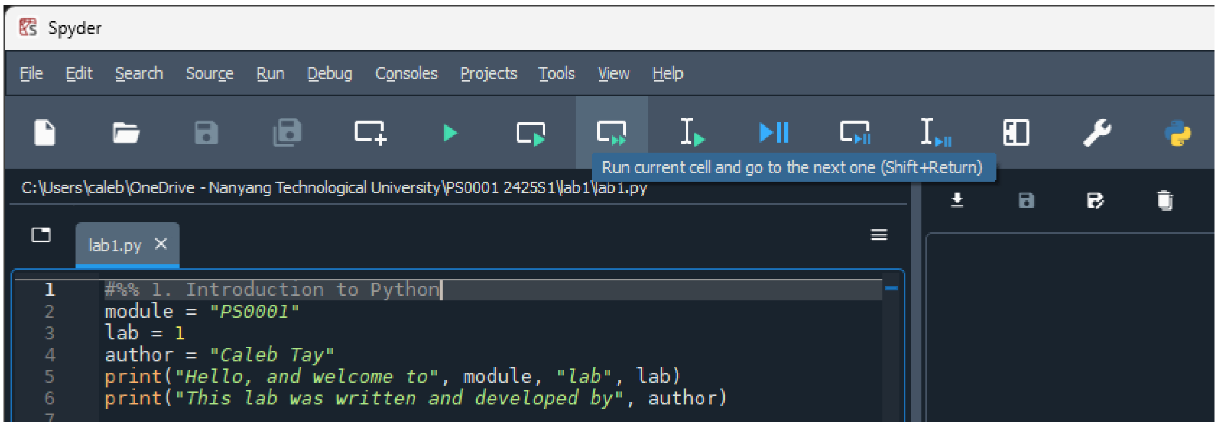
```
ython 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023,
3:26:23) [MSC v.1916 64 bit (AMD64)]
ype "copyright", "c      ts"   "license"    more inform   on.

Python 8.28.0 -- An                                     help.

n [1]:
```

Conda: miniconda3 (Python 3.11.5) ✓ LSP: Python Line 1, Col 1 UTF-8 CRLF RW Mem 49%

Figure 1: Take note of the five important elements. In future labs, we will go through the variable explorer and console in more depth.

In the variable explorer window, select the Variable Explorer tab, found along the bottom of that window (see the bottom of fig. 3).

Open this weeks code file in Spyder. You can do this through the menu bar, selecting File > Open. It should open in the file editor on the left. Navigate to the top of the file, **select the first line**, and click the ‘Run cell’ button shown in Figure 2.

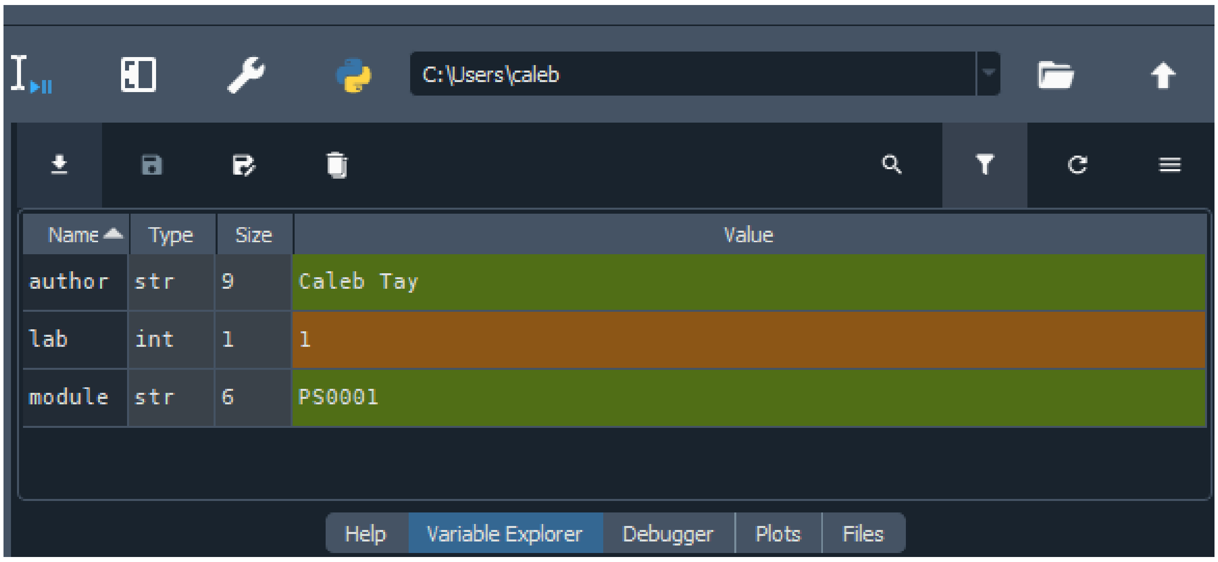

Spyder
File Edit Search Source Run Debug Consoles Projects Tools View Help
Run current cell and go to the next one (Shift+Return)
lab1.py

C:\Users\caleb\OneDrive - Nanyang Technological University\PS0001 2425S1\lab1\lab1.py

```python
#%% 1. Introduction to Python
module = "PS0001"
lab = 1
author = "Caleb Tay"
print("Hello, and welcome to", module, "lab", lab)
print("This lab was written and developed by", author)
```

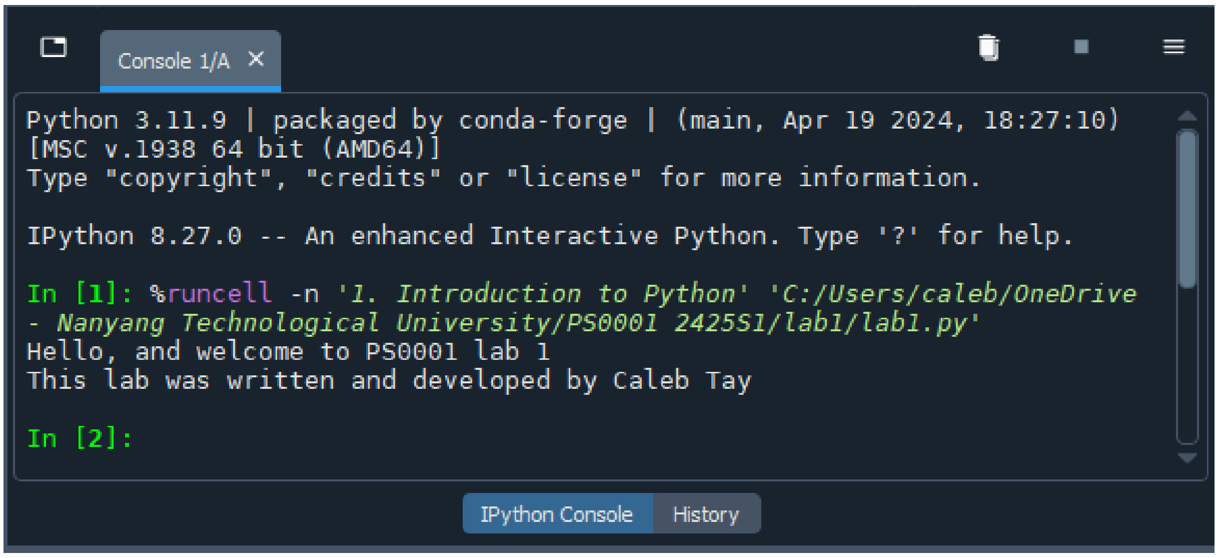
Figure 2: Run cell - hover and experiment with the buttons beside it, too.

Several things should have changed in Spyder (figs. 3 & 4). Some lines should have appeared below the cell, which is an *output* of the cell, and the variable explorer should now have 3 entries.



Name	Type	Size	Value
author	str	9	Caleb Tay
lab	int	1	1
module	str	6	PS0001

Figure 3: The variable explorer should have 3 entries.



```
Python 3.11.9 | packaged by conda-forge | (main, Apr 19 2024, 18:27:10)
[MSC v.1938 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runcell -n '1. Introduction to Python' 'C:/Users/caleb/OneDrive
- Nanyang Technological University/PS0001 2425S1/lab1/lab1.py'
Hello, and welcome to PS0001 lab 1
This lab was written and developed by Caleb Tay

In [2]:
```

Figure 4: The console should have entered a few lines, and displayed this text.

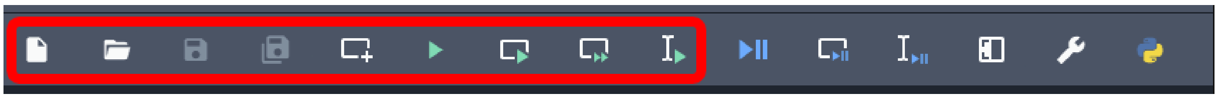


Figure 5: Experiment with the buttons in the box. Some of the buttons on the right will be covered in the future.

2.1 Code cells

What you have just run is a single *code cell* in a Python file, which are delimited by '#%%'. It ran all the lines from start to end in sequence, and performed exactly what it was told to do. It assigned 3 variables, and then printed 2 lines.

Code cells consist of several lines of code, meant to be run sequentially from start to end, and are a built-in functionality to Spyder. They may require further configuration in other applications (PyCharm, VSCode, etc.), and are mainly a convenience in these lessons to group code in a single file.

Hover over and experiment with the buttons shown in the box in Figure 5.

2.2 The `print()` function

Before we carry on, we must introduce to you the single most important function to beginning your Python journey: the `print()` function. This function will, for a while, be the sole way that the computer will be able to communicate with you.

Just as we are able to communicate with the computer by writing code, the computer can only do what it is told, and can only communicate with us when told to do so. The `print()` function, as the name suggests, prints to the console what it is told to print:

```python
print("Hello World!")
"I run, but I do not print!"
print("I take", "as many", "commas", "as you want!")
print("I can also print numbers:", 1, 3, 5, 7, 9)
```

When the above code block is run, *three lines* (not four!) are printed:

```
Hello World!
I can take as many commas as you want!
I can also print numbers: 1 3 5 7 9
```

Note that the second line was not printed. The computer recognised the statement - it is simply a *string* (more on this next time), and executed it. **Since you did not instruct the computer to print it, the computer didn't display anything to you!** Computers are dumb - they can only understand and do what it is told to do, and they must be told in a very particular, specific manner. **This particular, specific manner in which we write code to tell the computer what to do is called *syntax*.**

3 Variables

Variables are an essential concept in any programming language - they are how we tell the computer to remember some information, at least while running the current program. They are containers for information - in fact, you saw some variables in the very first code cell of this lab.

While variables are containers, we need to refer to them somehow, and we do so using *identifiers*. In Python, variable names (identifiers) **must** follow some rules:

- must begin with a letter or underscore;
- may contain letters, digits, and underscores;
- must not use Python keywords

'Letters' here, are extremely flexible, as you can see below, but we will continue to only use the English alphabet (A-Z) as letters for variable names.

```
变量_legal = 1
print("variable 变量_legal contains the value", 变量_legal)
```

Keywords in Python are *reserved* words that hold some special functionality and cannot be used as variable names. For the time being, we will leave it at this, though you can surely search what keywords Python has if you are interested.

Take note that variable names are *case sensitive*!

3.1 The *assignment* operator (=)

In Python (and many other languages), we can **assign** a value to a variable using the = symbol. This is your first bit of Python syntax - the simple assignment operator. The form is as follows, where LHS is a legal variable *identifier*, and RHS is an *expression*:

$$\text{LHS} = \text{RHS}$$

An *expression* is something that can be **evaluated** to a single value. As an example, in Python, we can do simple math - all of the following are legal expressions, because they can be evaluated. For example, `7 * 8` evaluates to `56`, and `9 / 10` evaluates to `0.9`.

```
3 + 4
5 - 6
7 * 8
9 / 10
```

Once we have associated a value with a variable, we can use the variables just as we would with the values that they contain. For example, the following two code blocks are equivalent:

```
hands = 2
print("It takes", hands, "hands to clap.")
```

```
print("It takes", 2, "hands to clap.")
```

You may interpret this as a 'substitution' of the *variable* for the *value* it contains.

Another thing to take note of is that we could have named the variables *anything we wanted*, as long as it is a valid name. The above code would also be equivalent to:

```
a = 2
print("It takes", a, "hands to clap.")
```

While equivalent to the computer - when reading the code, the variable name `a` carries much less meaning to the reader than the variable name `hands`. The simple idea here is to use the names of variables to convey meaning and information, so that when people read the code again, you are more easily able to understand the code. In these short scripts, perhaps meaningful variable names may not seem important, but as the length and complexity of the programs increase, meaningful names are important.

Variables are flexible containers - we can repeatedly change their value as we desire, and can hold any information that we want, as long as the right hand side is an expression. There is no limit to how many times a variable's values can change. Another educational example:

```
first = 3
second = 5
third = 7
second = first * third
third = second - 5
```

In the above code, executing line by line, we:

1. **assign** to a *variable* that we call `first` the *value* of 3.
2. **assign** to a *variable* `second` the *value* of 5.
3. **assign** to a *variable* `third` the *value* of 7.
4. **assign** to a *variable* `second` the *value* of the product of the *values* of `first` and `third`. This is an expression, because the computer knows what values `first` and `second` take on, and can 'substitute' them.
5. **assign** to a *variable* `third` the *value* of the subtraction of 5 from the *value* of `second`.

So, what are the values of `first`, `second`, and `third` at the end of each line? You could use Spyder's variable explorer, print out the values after each line to check, or track the change of each variable like in the table below.

line	first	second	third
`first = 3`	3		
`second = 5`	3	5	
`third = 7`	3	5	7
`second = first * third`	3	21	7
`third = second - 5`	3	21	16

As you can see, each line performs a specific action. In this script, the variables `first`, `second`, and `third` are repeatedly defined and updated.