TEMASEK
JUNIOR COLLEGE

Temasek Junior College
2022 JC1 H2 Computing
Practical 1 – Introduction to the Python IDLE

Session Objectives
By the end of this session, you will learn:
(i) what the Python IDLE is.
(ii) to interact with Python directly using the Python IDLE.
(iii) to write, edit and execute Python files with the Python IDLE.
(iv) to use the auto-indentation, code completion and call tip features of the Python IDLE
(v) what syntax errors and runtime errors are.

§1 What is Programming?

Computers work by following instructions. A collection of instructions for a computer is called a program. Programs are usually written by a programmer using a human-readable programming language such as Python. The human-readable form of a program is also called source code or just code for short. Programming languages also have strict rules for how instructions can be written such that source code is also readable to computers. To run or execute a program is to make a computer follow the instructions in the program. For the purpose of H2 Computing, we will be using Python for our programming section.

§2 What is the Python IDLE?

Every Python installation comes with an <u>I</u>ntegrated <u>D</u>evelopment and <u>L</u>earning <u>E</u>nvironment (IDLE). It is sometimes also known as an Integrated <u>D</u>evelopment <u>E</u>nvironment (IDE).

An IDLE or IDE
- is an application that performs the various stages of software design and implementation in a single integrated system.
- usually includes facilities for project management, design, graphical design, programming, testing and producing documentation.

Some IDLEs or IDEs also
- enable increased automation e.g. simple designs can be automatically converted into program code.
- perform version management to ensure that the correct (usually the latest tested) version of each module are used in each version of software produced.

The Python IDLE consists of
- an interactive interpreter (**see Section 2.1**)
- a file editor (**see Section 2.2**).

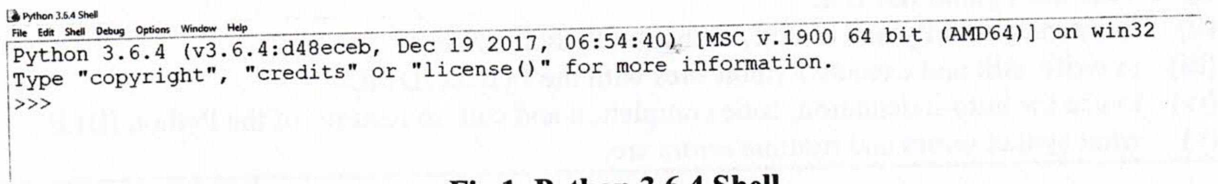
You can type code into both the interactive window and the editor window. The difference between the two windows is in how they execute code.

In this practical session, you'll learn how to execute Python code in both windows.

§2.1 Python Interactive Interpreter

§2.1.1 Introduction to Python Shell

Upon starting the Python IDLE, a window known as the Python Shell is launched **(see Fig 1)**. It is an interactive text-based user interface for interacting with your computer in the Python language.



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Fig 1. Python 3.6.4 Shell

The Python Shell contains some text that provides information about the version of Python that your IDLE is running. For H2 Computing, ensure that you are running Python 3.6.4.

You can also see information about your operating system and some commands that you can type into the Python Shell to get help and view information about Python.

The >>> symbol on the last line is called the prompt. You can type a bit of Python code after the prompt and then press "Enter" to run the code. The results of running the code that you have typed will be displayed immediately. Hence the idea of interactive.

Exercise 1
In the Python Shell, type in the each of the following and press "Enter". Observe the results that show.
(a) copyright
(b) credits
(c) license()

§2.1.2 The Read-Evaluate-Print Loop (REPL)

Exercise 2
In the Python Shell, type and run the expression 1 + 1. Observe the results that show.

In **Exercise 2**, notice that the Python Shell evaluates the expression 1 + 1. It then displays the result 2. Thereafter, it displays another prompt.

In fact, what you have observed can be describe as a loop with 3 steps:
(1) Python will **read** the code entered at the prompt.
(2) Python will **evaluate** the code
(3) Python will **print** the result and wait for more input.
This loop is known as the **read-evaluate-print loop (REPL)**. As such, the Python Shell is also known as the Python REPL, or just "REPL" for short.

§2.1.3 The "Hello World" Rite of Passage

Exercise 3
A rite of passage for every human being learning to code is to write a program that prints "Hello, World" on the screen.

In the REPL, type and run the following code:

print("Hello, World")

You have just used the print( ) **function** in **Exercise 3**. Let us understand how the code works:

(1) The pair of parentheses ( ) instructs Python to activate the print function.
(2) The pair of parentheses ( ) also encloses everything that gets sent to the function as input.
(3) The pair of double quotation marks " " indicate that Hello, World is just text. A pair of single quotation marks ' ' can also be used as an alternative.

In programming, a **function** is a sequence of code that is given an identifier (a name given to the function) and returns a single value. We shall learn more about functions in future practical sessions.

You should have by now realised that the REPL displays different parts of your code in different colours. This feature makes it easier to identify different parts of a code. By default, a function is displayed in purple and text in green.

For H2 Computing, we will stick to the default colour scheme for consistency.

§2.1.4 Restarting the REPL

There is no need to close and restart the REPL everytime. You can simply select "Shell" and click on "Restart Shell" **(see Fig 2)**. Alternatively, you may press the keys "Ctrl" and "F6" simultaneously.

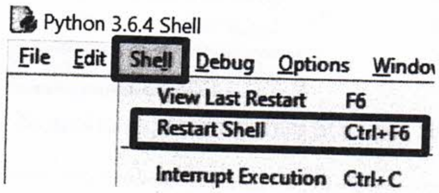
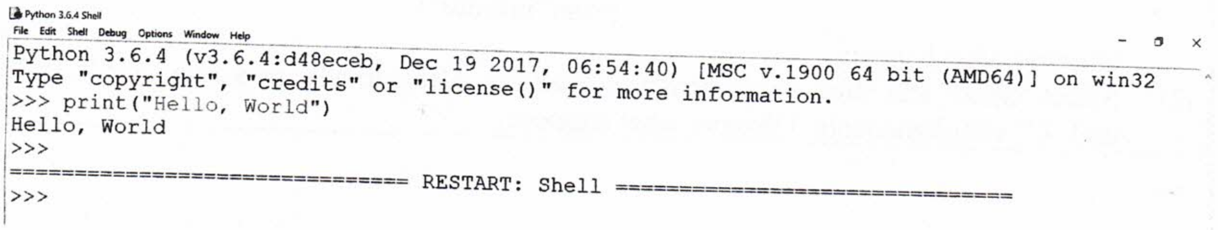


Fig 2. Restarting the REPL

Upon successfully restart, you will see the line "RESTART: Shell" **(see Fig 3)**.



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello, World")
Hello, World
>>>
============================ RESTART: Shell ==============================
>>>
```

Fig 3. Successful Restart of REPL

Notice from the screen capture that the previous lines were not cleared from the window. What has actually happened is that a new "instance" of the REPL has been activated and everything that was previous previously entered, processed and stored has been "forgotten".

Exercise 4
Perform the following steps in the REPL:
(1) Type and run x = 5.
(2) Type and run print(x). Observe what happens.
(3) Restart the REPL using by selecting "Shell" and clicking on "Restart Shell" or using keys "Ctrl" and "F6" simultaneously.
(4) Type and run print(x). Observe what happens.

Let us understand what has occurred in **Exercise 4**:
(1) In Step 1, the value of 5 was assigned to x.
(2) In Step 2, printing x will result in the printing of the value that was assigned to x i.e. 5.
(3) In Step 3, restarting the REPL will result in everything that was previously entered, processed and stored to be "forgotten". Hence the value of 5 that was assigned to x has been "forgotten". As such, x is no longer defined.
(4) Hence in Step 4, printing x results in an error message that says that x is not defined. (We shall learn more about the error that you have seen future practical sessions.)

§2.1.5 Interrupting an Execution of Code

You can also interrupt the execution of a code in the REPL by selecting "Shell", then clicking on "Interrupt Execution" (**see Fig 4**). Alternatively, you may press the keys "Ctrl" and "C" simultaneously. Both methods results in a "KeyboardInterrupt" and stops the execution of the code.

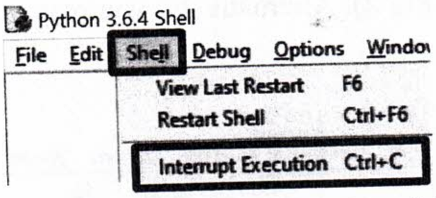


Fig 4. Interrupting an Execution of Code

Exercise 5
Perform the following steps in the REPL:
(1) Type and run the following code:

```
while True:
    print("infinite")
```

Observe what happens.
(2) Select "Shell" and click on "Interrupt Execution". Alternatively, press the keys "Ctrl" and "C" simultaneously. Observe what happens.

§2.2 Python File Editor

§2.2.1 Introduction to File Editor

The REPL executes a single line of code at a time. This is useful for trying out small code snippets and exploring the Python language. What happens then if you want to execute many lines of Python code that form a program all at once?

To type many lines of code at once, you will need to launch the File Editor from within the REPL. This can be accessed by selecting "File" and clicking on "New File" (see Fig 5 and 6).

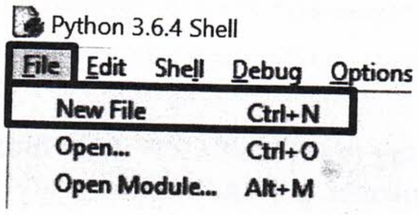


Fig 5. Launching the File Editor from within REPL

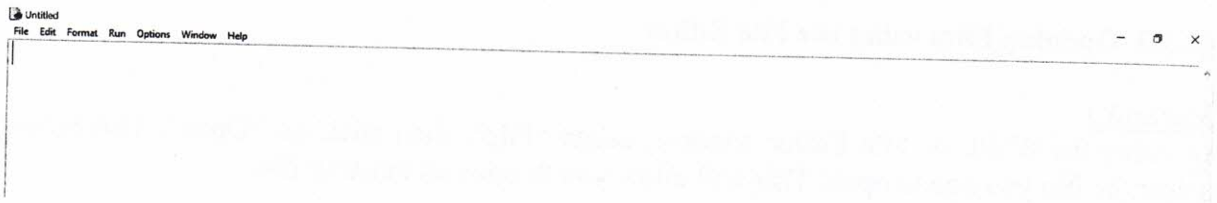


Fig 6. Python File Editor

Notice that the REPL remains opened. It will display the results generated by the code that you type in the File Editor. Hence it will be useful to arrange the REPL and File Editor side by side so that you can see both windows concurrently.

§2.2.2 Coding, Saving and Executing Programs in the File Editor

Exercise 6
Perform the following steps in the File Editor:
(1) Type the following code:

```
print("Hello, World")
```

(2) Select "File", then click on "Save".
(3) Save the file as **hello_world.py** on your Desktop.
(4) Select "Run", then click on "Run Module". Alternatively, press the key "F5".
(5) Observe what happens in the REPL window.

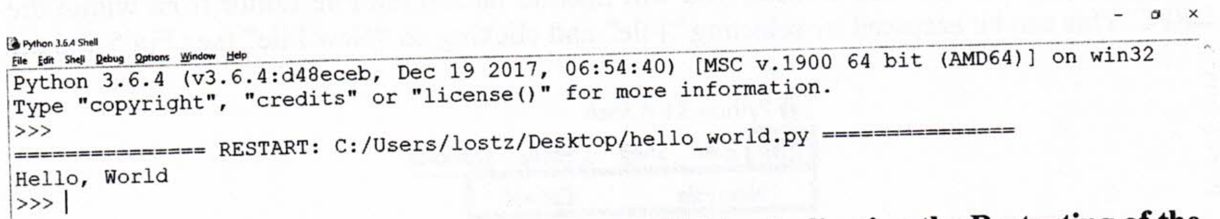
Let us understand what has occurred in **Exercise 6**:

(1) In Step 1, you were typing in the code print("Hello, World"). Notice that the
- File Editor does not contain the >>> prompt. There is no need to include >>> either.
- Same colour scheme as with the REPL applies for different parts of the code in the File Editor.

(2) Steps 2 and 3 are necessary steps if you are coding a program using a File Editor and want to run it. The .py extension indicates that a file contains Python code. To retain the colour scheme for different parts of the code, it is necessary for the .py extension to be used.

(3) Performing Step 4 allows you to run the program that has been coded and saved in the File Editor currently. The Python Interpreter, which is the program that actually executes the code will be restarted. This ensures that the same program will always behave the same way each time its code is executed **(see Fig 7).**

(4) The results of the program executed in the File Editor will always appear in the REPL window **(see Fig 7).**



```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:/Users/lostz/Desktop/hello_world.py ===============
Hello, World
>>> |
```

Fig 7. REPL Window Showing the "RESTART" Line Indicating the Restarting of the Python Interpreter to Run the hello_world.py Program

§2.2.3 Opening Files using the File Editor

Method 1
In either the REPL or File Editor window, select "File", then click on "Open". Thereafter, select the file you one to open. This will allow you to open an existing file.

Method 2
Perform a right-click on the file which you wish to open, then select "Edit with IDLE".

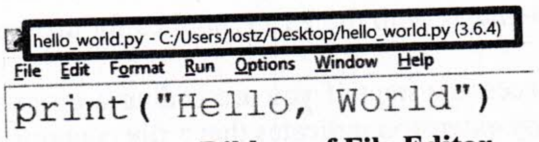
For both methods, a new File Editor window will open for each file. Hence you can open several files concurrently.

Note
Double-clicking on a .py file executes the program with the system Python. A program window will appear and then disappear immediately after the program terminates. The process is almost instantaneous and completes before you can even see any visible results. For now, the best way to run your Python programs is to open them using Method 1 or 2, then select "Run" and click on "Run Module".

§2.2.4 Editing Files using the File Editor

The top ribbon of the File Editor contains three pieces of important information **(see Fig 8):**

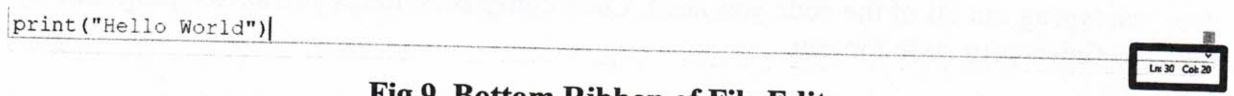
(1) File name
(2) File path i.e. the location of the file you are editing
(3) Python version


hello_world.py - C:/Users/lostz/Desktop/hello_world.py (3.6.4)
File Edit Format Run Options Window Help

```
print("Hello, World")
```

Fig 8. Top Ribbon of File Editor

The bottom ribbon of File Editor also contains two values (**see Fig 9**):
(1) **Ln**: the line number of the code which your cursor is on
(2) **Col**: the column number that your cursor is on



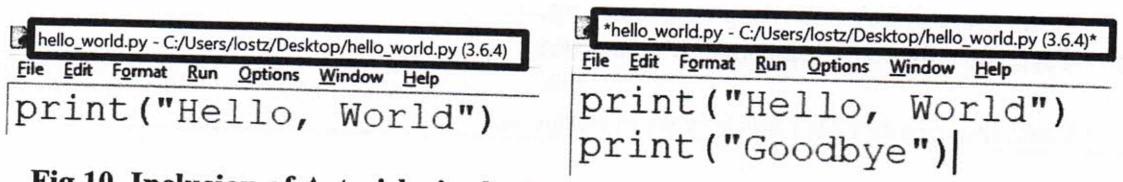
```
print("Hello World")
```

Fig 9. Bottom Ribbon of File Editor

These two values are also found in the REPL. They are useful to help you locate errors quickly. In addition, they help you make sure that you stay within a certain line width e.g. all lines to be limited to a maximum of 79 characters based on the PEP 8 style. (We shall learn more about the PEP 8 style in future practical sessions.)

There are a few visual cues in the File Editor that will help you remember to save your work. If you were to observe closely, you will notice that the File Editor uses asterisks * in the top ribbon to indicate unsaved changes.

In the figure below (see **Fig 10**), the left panel shows the hello_world.py program upon opening and the right panel shows the program with an additional line being coded but before the changes were saved. Notice that the the description in the top ribbon of the right panel is now bounded by a pair of asterisks *. This isn't seen in the top ribbon of the left panel when the file was just opened.



hello_world.py - C:/Users/lostz/Desktop/hello_world.py (3.6.4)

```
print("Hello, World")
```

hello_world.py - C:/Users/lostz/Desktop/hello_world.py (3.6.4)

```
print("Hello, World")
print("Goodbye")
```

File Edit Format Run Options Window Help
File Edit Format Run Options Window Help

Fig 10. Inclusion of Asterisks in the Top Ribbon Indicating Unsaved Changes

§2.3 Useful Features

The Python IDLE offers a few features that you'll see in most professional IDEs to help you code faster. These features include automatic indentation, code completion and call tips.

§2.3.1 Automatic Indentation

The Python IDLE will automatically indent your code when it needs to start a new block. This usually happens after you type a colon (:). When you hit the enter key after the colon, your cursor will automatically move over a certain number of spaces and begin a new code block.

You can configure how many spaces the cursor will move in the settings. For H2 Computing, we will keep to the default of four spaces. This is in agreement with the standard style for well-written Python code as agreed by the developers of Python.

The standard style was formalised as the PEP 8 and includes rules on indentation, whitespace and more.

§2.3.2 Code Completion

When you're writing code for a large project or a complicated problem, you can spend a lot of time just typing out all of the code you need. Code completion helps you save typing time by trying to finish your code for you.

Python IDLE has basic code completion functionality. It can only autocomplete the names of functions and classes. To use autocompletion in the editor, just press the tab key after a sequence of text.

If the sequence is unique to a particular function or class, the name will be automatically completed e.g. pressing the tab key after the sequence of text "pri" will autocomplete the name "print". This is because no other names of functions and classes begin with the sequence "pri"

If the sequence of text appears in more than one function or class, a list of possible names pops up for selection e.g. pressing the tab key after the sequence of text "ma" will bring up a list of possible names of functions and classes beginning with the sequence "ma".

Exercise 7
Perform the following steps in the File Editor or REPL:

(1) Type the following sequence of characters:

pri

Press the Tab key and observe what happens.

(2) Type the following sequence of characters:

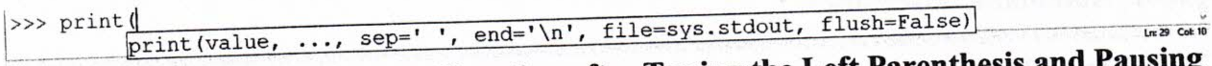
ma

Press the Tab key and observe what happens.

§2.3.3 Call Tips

Python IDLE will also provide call tips. A call tip is like a hint for a certain part of your code to help you remember what that element needs. After you type the left parenthesis to begin a function call, a call tip will appear if you don't type anything for a few seconds.

For example, if you can't quite remember how print a statement, then you can pause after the opening parenthesis to bring up the call tip **(see Fig 11)**.



```
>>> print(
        print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Fig 11. Call Tip for the print() Function after Typing the Left Parenthesis and Pausing

Note
For H2 Computing, code completion and call tips may not be available depending on the nature of the assessment. It is best to know all the Python code that will be assessed.

§2.4 Syntax and Runtime Errors

Making mistakes in coding is unavoidable and this can happen even to the most expert of programmers. Mistakes in programs are called **errors**. In this section, we shall explore two main types of errors: **syntax errors** and **runtime errors**.

§2.4.1 Syntax Errors

A **syntax error** occurs when code statements cannot be understood because they do not follow the rules laid down by the programming language. (We shall learn more about the different types of syntax errors as we progress through the course of study).

Exercise 8
In the File Editor, type the following code, then save it as hello_world.py and run it:

```
print("Hello, World)
```

Observe that in **Exercise 8**, the code cannot be executed. Instead, the "SyntaxError" popup that contains the line "EOL while scanning string literal" appears.

There are two terms in this message that may be unfamiliar:

(1) A string literal is text enclosed in quotation marks. "Hello, World" is a string literal.
(2) EOL stands for end of line.

The message thus tells you that Python got to the end of a line while attempting to read a string literal.

In the rules laid down for Python programming, string literals must be terminated with a quotation mark before the end of a line.

In the File Editor, the erroneous line is indicated with a red bar at the end. This helps you quickly locate the line of code with the syntax error. In **Exercise 8**, everything after the opening quotation mark, including the closing parenthesis becomes part of a string literal in the absence of a closing quotation mark.

§2.4.2 Runtime Errors

A **runtime error** occurs during program execution if a mistake is made in the processing algorithm or as a result of external effects not catered for by the program, such as lack of memory or unusual data.

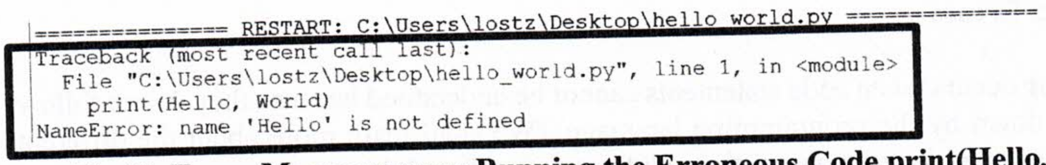
While the Python IDLE is able to detect syntax errors before a program starts running, it can only detect runtime errors only when a program is running.

Exercise 9
In the File Editor, type the following code, then save it as hello_world.py and run it:

```
print(Hello, World)
```

Observe that in **Exercise 9**, the colour of the text Hello World is black instead of green. Try typing print("Hello, World") to make a comparison. This means that Hello World is no longer recognised as text. Recall that the default colour scheme indicates text as green.

When the code is executed, we observe that the following error messages were displayed in the REPL **(see Fig 12)**.



```
=============== RESTART: C:\Users\lostz\Desktop\hello_world.py ===============
Traceback (most recent call last):
  File "C:\Users\lostz\Desktop\hello_world.py", line 1, in <module>
    print(Hello, World)
NameError: name 'Hello' is not defined
```

Fig 12. Error Messages upon Running the Erroneous Code print(Hello, World)

Whenever a runtime error occurs, Python stops executing the program and several lines of error messages called a **traceback** is displayed.

The traceback shows useful information about the error. Tracebacks are best read from the bottom up:

- The **last line** of the traceback tells you the name of the error and the error message. In this case, a NameError occurred because the name Hello is not defined anywhere.
- The **second last line** shows you the code that produced the error. There's only one line of code in hello_world.py, so it's not hard to guess where the problem is. This information is more helpful for larger files.
- The **third last line** tells you the name of the file and the line number so you can go to the exact spot in your code where the error occurred.

__Tutorial 1__
Complete the following questions and submit on Google Classroom.

1) Write 2 programs that won't run because it has a syntax error.
__**Requirements**__
The cause of syntax error cannot be due to missing opening or closing quotation marks.
The cause of syntax error in each program must be different.

Name the 2 programs "SE1.py" and "SE2.py" respectively.

2) Write 2 program that crashes only while it's running because it has a runtime error.
__**Requirements**__
The cause of runtime error cannot be due to a pair of missing quotation marks.
The cause of runtime error in each program must be different.

Name the 2 programs "RE1.py" and "RE2.py" respectively.

3) Is it possible to write a program with both a syntax error and a runtime error and have both of them detected when the program is executed? Why or why not?