Raspberry Pi for Beginners

LESSON 5

MAKERHOUSE
EMPOWERING MAKERS

TABLE OF CONTENTS

GETTING STARTED WITH PYTHON BASICS

1 INTRODUCTION

We were having been introduced with Python as early in Lesson 3 -- Python programming with Turtle. Noticed that, Python is easy to learn and use – modest, little time and effort to start, plus the syntax is readable – great for a first programming language to learn. People with low or zero knowledge of programming can easily pick up quickly due to its simplicity.

Did you agree on this after first experienced -- controlling Turtle to do whatever you want in lesson 3 -- with Python?

Moreover, Python is broadly adopted and supported, where we can see it very popular and widely used – building professional quality software – standalone or web services.

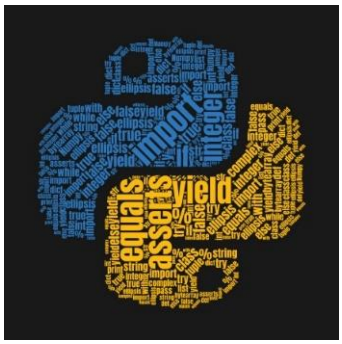

Figure 1: Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.

And for today, the lesson will be different from Lesson 3. If in Lesson 3 we directly use the Turtle module in Python to do drawings based on arts, computer sciences and maths but today lesson, we will be focused on Basics of Python language – which we didn't detail explained in Lesson 3.

Are you ready? Let's go!

1.1 GETTING STARTED

Still, remember Terminal?

Even though Python doesn't specifically require unique integrated development environment (IDE), we will be entirely using Terminal (Itxterminal) for interactive Python console, create files and folders, write a script and to executes it.

With Great Power, Comes Great Responsibility!

2 PYTHON BASICS

This lesson will be emphasized on the basics only – each step you take reveals new horizon, you have taken the first step today!

2.1 WHICH PYTHON?

It is the most popular programming language thus inspired the name of Raspberry **Pi**. There are two version of Python pre-installed on Raspbian OS – Python 2 and Python 3. You can use both – scripts in this lesson will more likely use Python 3

2.2 PYTHON CONSOLE

In lesson 3, we were using Python console (interactive) – find a helps on specific Turtle Python instruction.

We were typing python (Python 2) on Terminal, remember?

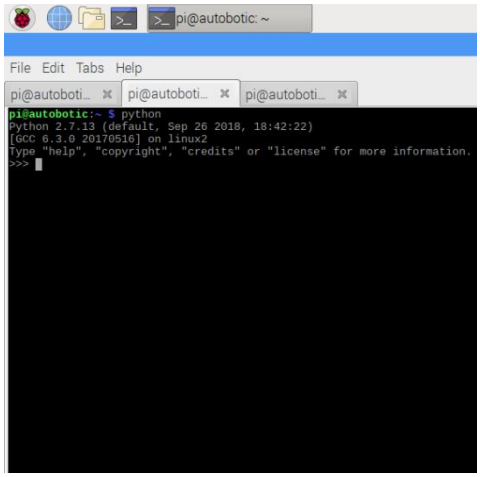

Figure 1: In terminal, type: python for Python 2

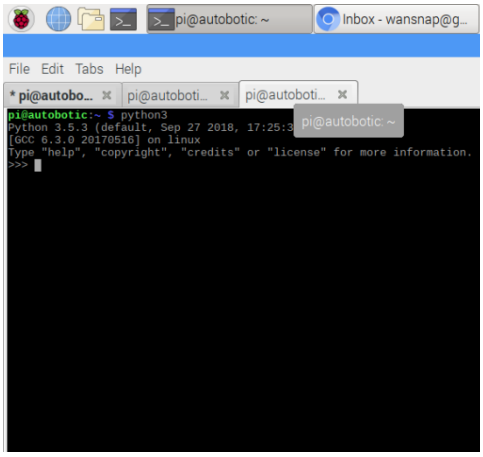

Figure 2: In terminal, type: python 3 for Python 3

Python console is handy. It is used to write a short Python command or for troubleshooting.

2.3 CREATE A FILE AND FOLDER FOR PYTHON SCRIPTS

1. Create folders – **Scripts**
    a. **mkdir** Scripts
2. Create file – **hello_world.py**
    a. **touch** hello_world.py
3. Edit file – **hello_world.py**
    a. **nano** hello_word.py
    b. **print("Hello, World!")**
    c. **ctrl + x > y > Enter (Return)**

***** Requiring understanding Lesson 4: Mastering the Command Line***

2.4 EXECUTE PYTHON SCRIPTS

Executing Python scripts on Terminal is very straight forward. Just type python (or python3) and following with programs name – python hello_world.py or python3 hello_world.py.

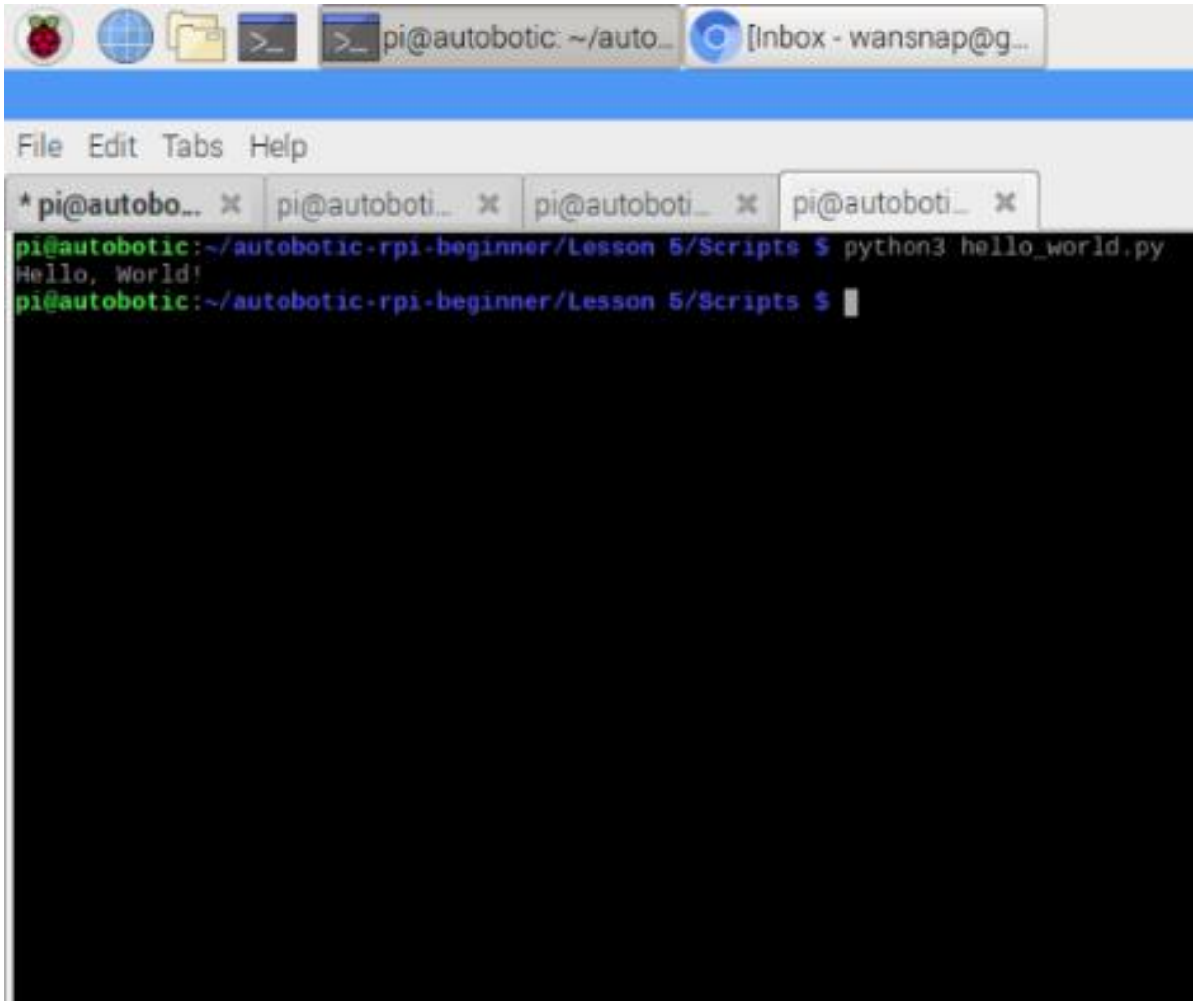

Figure 3: Executing Python 3 hello_world.py

2.5 VARIABLES

It is very useful in Python (every programming language actually). Used to assign to a value for later and elsewhere use. Since *Python executes instruction in orderly – variables need to be declared before can be used!*

Your variable can have almost any name, however, shall start with letter or underscore. Importantly, it cannot have same name as Python reserved keywords.

Reserved Python Keyword										
False	class	finally	is	return	assert	break	as	yield	global	pass
None	continue	for	lambda	try	else	except	elif	and	not	pass
True	def	from	nonlocal	while	import	in	if	del	with	or

Table 1: Reserved words in Python -- cannot use a keyword as variable name, function name or any other identifier.

Recommendation: use lowercase and separated with underscore – easy to maintain plus help others to understand your code well.

Variable doesn't require specific type of value to be assign – Python will auto be recognized it. Basically, the variable type may consist of logical /bool (True/False), string ("any_string" or 'anything') and numbers (float and integer). To check these kinds of variable value type – use type function – **type**(variables) or **type**(values).

```
a = 123
b = 12.34
c = "Hello"
d = 'Hello'
e = True
```

2.6 DISPLAYING OUTPUT

Displa ying an output can be achieved in Python with the **print** function – **print**(variable).

```
#!/usr/bin/env python3
print("Hello, World!")
```


2.7 READING USER INPUT

An **input** function a made readily in Python for received/acquire user input – **input**("Ask input").

```
#!/usr/bin/env python3
x = input("Enter Value: ")
print(x)
```

***** Acquiring user input in Python used raw_input function instead of input***

2.8 ARITHMETIC

An arithmetic process can be done in Python. Numbers – integer or float – can undergoing the arithmetic process without any problem – simply implement the process directly.

No	Arithmetic	Description
1	+	Addition
2	-	Subtraction
3	*	Multiplication
4	/	Division
5	%	Modulus – remainder
6	**	Power

```
#!/usr/bin/env python3
tempC = input("Enter temp in C: ")
tempF = (int(tempC) * 9) / 5 + 32
print(tempF)
```

Table 2: Python support arithmetic processes

2.9 STRINGS

One of type variables in Python – strings.

1. **Create a Strings**:

Strings in Python can be easily recognized if it is **encapsulated with double/single quotes marks**.

```
#!/usr/bin/env python3
s = "abc def"
print(s)
```

***** escape characters -- to include a tab, use \t, and for a newline, use \n***

2. Concatenate Strings:

Strings in Python can be concatenate (join) by using "**+**" between two strings; **string = str + ing**.

```
#!/usr/bin/env python3
s = "abc" + "def"
print(s)
```

2.10 COVERTING NUMBER TO STRING

With the **str** function, number data types can be simple converted to string type; **str**(variables)

```
#!/usr/bin/env python3
s = str(123)
print(s)
```

2.11 CONVERTING STRING TO NUMBER

In order to convert string data types to number data types can use **int**(string) for integer and **float**(string) for floating point.

```
#!/usr/bin/env python3
s = int("-123")
print(s)
```

2.12 STRING LENGTH

To count/find the total character used in strings – **len**(string) function.

```
#!/usr/bin/env python3
s = "abc def"
print(len(s))
```


2.13 FIND STRING IN STRING

To find the position of one string within another use string.**find**(string)

```
#!/usr/bin/env python3
s = "abcdefghi"
print(s.find("def"))
```

2.14 EXTRACT PART IN STRING

To cut out a section of a string between certain character positions – Python use "**[:]**" notation

```
#!/usr/bin/env python3
s = "abcdefghi"
print(s[1:5])
```

2.15 REPLACE STRING IN STRING

To replace all occurrences of a string within another string: string.replace("old", "new")

```
#!/usr/bin/env python3
s = "It was the best of X. It was the worst of X"
s.replace("X", "times")
print(s)
```

2.16 UPPER/LOWER CASE

To convert all the characters in a string to upper/lowercase letters: string.upper(), string.lower()

***** you can change accordingly. Don't be afraid to try!***

```
#!/usr/bin/env python3
s = "aBcDe"
print(s.upper())
```


2.17 CONDITIONAL COMMAND

Run some Python commands only when some condition is true: **if, if…else, if…elif…else**

```
#!/usr/bin/env python3
x = 101
if x > 100:
        print("x is big")
```

***** if – condition***

```
#!/usr/bin/env python3
x = 101
if x > 100:
        print("x is big")
else:
        print("x is small")
print("This will always print")
```

***** if -- else – condition***

```
#!/usr/bin/env python3
x = 90
if x > 100:
        print("x is big")
elif x < 10:
        print("x is small")
else:
        print("x is medium")
```

***** if – elif -- else – condition***

2.18 COMPARING VALUES

To compare values with each other:

No	Comparison	Description
1	<	Less than
2	>	Greater than
3	==	Equal to
4	<=	Less than or equal to
5	>=	Greater than or equal to
6	!= or <>	Not equal to

Table 3: Python support arithmetic processes

```
#!/usr/bin/env python3
print(1 != 3)
print("aa" > "ab")
```

**** strings are compared lexicographically**

2.19 LOGICAL OPERATORS

To specify a complex condition in an **if** statement – **and**, **or**, and **not**.

```
#!/usr/bin/env python3
x = 17
if x >= 10 and x <= 20:
        print("x is in the middle")
```

2.20 REPEATING INSTRUCTION AN EXACT NUMBER OF TIMES

Repeat some program code an exact number of times – **for** – will iterate over a range:

```
#!/usr/bin/env python3
for I in range (1, 11):
        print(i)
```

2.21 REPEATING INSTRUCTION UNTIL SOME CONDITION CHANGE

Repeat some program code until something changes – **while** – repeats nested commands until its condition becomes False.

```
#!/usr/bin/env python3
answer = " "
while answer != "X":
        answer = input("Enter command: ")
```


2.22 BREAKING OUT OF LOOPS

Exiting the loop if some condition occurs

```
#!/usr/bin/env python3
while True:
        answer = input("Enter command: ")
        if answer == "X":
                break
```

2.23 DEFINING A FUNCTION

To avoid repeating the same code over and over in a program.

```
#!/usr/bin/env python3
def count_to_10 ():
        for i in range (1, 11):
                print(i)
count_to_10 ()
```

***** function without parameters***

```
#!/usr/bin/env python3
def count_to_n (n):
        for i in range (1, n + 11):
                print(i)
count_to_n (10)
```

***** function with parameters – required***


```
#!/usr/bin/env python3
def count_to_n (n = 10):
        for i in range (1, n + 11):
                print(i)
count_to_n ()
```

***** function with parameters – optional***

```
#!/usr/bin/env python3
def count_to_n (from_num = 1, to_num = 10):
        for i in range (from_num, to_num + 1):
                print(i)
count_to_n ()
count_to_n (2)
count_to_n (2, 8)
```

***** function with multiple parameter – optional***

```
#!/usr/bin/env python3
def make_polite (sentence):
        return sentence + "please"
print(make_polite("Pass the cheese"))
```

***** function with sentence as parameter***

