PYTHON BEGINS

An Introduction To Python Programming

RAHUL . J . ROY

PYTHON BEGINS

AN INTRODUCTION TO PYTHON PROGRAMMING

BY

RAHUL.J.ROY

INDEX

What is python?

If you are a beginner to the Programming field, then the first thing that comes in your mind would be “Python is a kind of Snake”.
Of course, real life Pythons are not such cute as I have shown and also Python that we will be studying is not the Actual Python (Snake).
Before we get into the more technical support, let's just say python is a programming language. It is easy to understand.

What is a programming language?

- As I could only hope that you are a human. We humans generally use language to communicate with each other. Similarly, when a computer wants to communicate, what will be the language they will use?
- It's obvious that they don't talk with the help of human language.
- So developers use a programming language such that a set of instructions could be used in such a way that a computer or say an application of a computer works properly.

So, now we can start our journey of Python Programming. Nowadays, Python programming has a number of definitions like it is an Interpreted, Object Oriented, General-Purpose language. Whoaa !! That's a lot of technical words to understand. Thus, for now, Consider Python as a programming language which can be used to make applications for a particular system. If defined more complexly, it can be used in the creation of artificial intelligence models, machine learning models and so on. Let's just not concentrate on that much.

But, why should we learn python?

- First and foremost, it's open sourced which means it's free to use.
- Secondly, it is very easy to understand as most of the python language has keywords which are easily used in normal English language.
- It's application has grown very large in the last few years.
- Also, it's code is not much big as compared to the other programming languages.
- Most of the artificial Intelligence and Machine Learning models and applications are generally created using Python.

I hope now you might get an idea of what is somewhat python programming. So, let's Get Started

Lesson 1: Variable

- Hearing the word "variable" what's the first thing that comes to your mind, huh?
- Relax, Don't put so much pressure on your brain, I'll explain.
- I hope you have a name.
- Surprisingly, I also have a name.
- Hi, my name's Roy.
- But why am I telling you this right now?
- It's because variables are nothing but a way to name someone.
- In technical language, Variables are used to store some data which will be stored in your memory space of the system.
- So, now you get an idea what a variable is. **A variable can be used to store data.**
- For example, suppose you are solving mathematics in your school. For those who don't like mathematics, don't worry, I am taking an example of a 2nd grade student. You are solving an equation, there must be some value in the equation.
- Say (x has a value of 9) and (y has a value of 10). Here, you want to multiply them and get the answer.
- Don't go on multiplying them, just see x and y, they are the variables in this case.

Syntax (It generally means a rules to define or say grammar of programming)

```
variable_name = 'its_value'
```

Example:

```
my_name = 'Roy'
Current_age = 20
```

Here **my_name** and **Current_age** are the two **variables** and their **values** are '**Roy**' and **20**.

You see how easy it is to understand the meaning of a variable. But there is something more to it, like the rules for naming a variable. Why are there rules for naming? Say like in the real world, there is a name for everything for a reason. Can we call a dog a cat? Of Course not! Because, we have named dogs as dogs already (I don't know who first called a dog a dog). Just like that there are some rules for naming a variable.

Rules for naming a variable:

1. A variable name must not start with a number.
2. A variable name must start with a letter or underscore(_).
3. A variable name must contain only alphanumeric values (**Aa - Zz**, **0-9** , _).
4. A variable name should not be a python reserved keyword. An example of a reserved keyword is **print.**
5. Variable names are case sensitive which means: (**Name**, **name**, **NAME**) These three are three different variables.

Apart from all these rules, if you try to do differently, then the python compiler will cause an error.
We can define multiple variables in a single line.

Syntax

variable_1, variable_2, variable_3 ... variable_n = value_1, value_2, value_3 .... value_n

Example

number1, number2, number3 = 1,2,3
Also, we can define the same value to multiple variables.

Example

X = Y = Z = 7

Some of the valid and invalid variable examples:

```
1st_name = 'Amelia'
last name = 'Smith'
class = 'seventh'
pass 'yes'
```

These are the **invalid** variables.
In the first variable declarations, we used number (1) to start the variable name which is not allowed.
Then, in second variable declaration, we used space in the variable which is also not allowed. In the third variable declaration, we used a python keyword (class) to name a variable which is prohibited.
And in the last variable declaration, we used Python keyword (pass), also we didn't used assignment operator (=) in between the variable name and its value.

To solve the above invalid variable declarations, we will use valid variables as follows below:

```
first_name = 'Amelia'
last_name = 'Smith'
student_class = 'seventh'
is_student_pass = 'yes'
```

These are the **valid** variables:
In the first variable declaration, we use a proper alphabetical to start the variable name.
In the second variable declaration, we use an underscore (_) instead of space.
In the third variable, we used a prefix as student such that, it does not clash with the Python keyword (class). Similarly, in the last variable declaration, we also use a prefix and also given the assignment operator in between the variable name and its value.

Lesson 2: print()

We understood what's a variable in programming, but what is this print now then? Say we define Variables earlier, so what if we want to show the values of the variables to someone or just if we want to display a set of messages or values of variables we use the print function.
Print Function is used to display messages or can be used to show output data.
If print Function consists of a message within a quote (",') then the message will be displayed as it is mentioned.

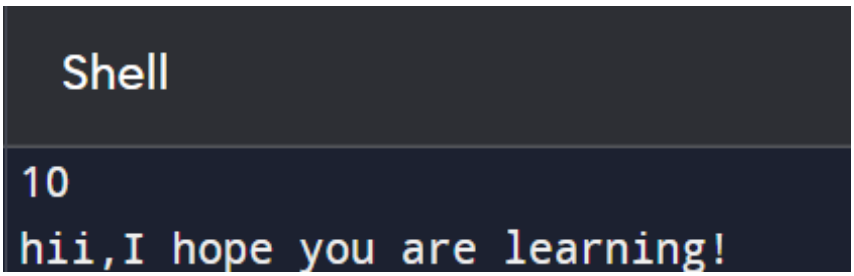
Syntax:

print(object(),sep=separator)
Here **object()** could be anything like a string message or a value of a variable or a variable.
sep is a separator which is optional which can be used to separate object's, if there exists more than one object.

Example:

```
print(10)
print('hii,I hope you are learning! ')
```

Output window



The above output can be seen when you run the program.

I'll be asking you number of questions in between the topics so that you will be able to know you well you have been understanding till now.

Q.1 What will be the output of the following statement:

```
Print (" It's obvious ")
```

(a) `It's obvious`
(b) `it's obvious`
(c) `IT'S OBVIOUS`
(d) `NameError: name 'Print' is not defined`

I'll explain the answers later stages as I don't want you to feel like you've been doing well or doing bad.

Let me show you now how you define a variable and then print it's value to the screen.

```
Name = 'Thor'
print("Name = ",Name)
```

Output Window

```
Name =  Thor
```

In the above example you see I have declared a variable named **Name** and has a value as **Thor**. Then, I wanted it to show the name with a message showing that the name is equals to Thor so I have written (Name =) in quotes such that it will be displayed as it is. Then I used the separator(,) to separate the two objects (the message and the value of variable). Finally just write the variable name to show its value.

Answer 1
So Have you guessed your answers for Q.1?
If you have answered option(a),Then, I must say you have been proved wrong on the very first question I asked with multiple choices. But, if you have guessed option(d) then I must say you are really concentrated. But why? I just said before that if a print Function consists of a message within a quote, then the message will be displayed as it is. Then why didn't it print as it is, it's because I have also said that python is case sensitive which means print() and Print() are not the same. So that's the reason it will cause an error to the screen.

Now it's time for the next question

Q.2 What will be the output of the following statement:

```
age = 20
print("Age =","Age")
```

(a)
```
Age = 20
```

(b)
```
age = 20
```

(c)
```
AGE = 20
```

(d)
```
Age = Age
```

Lesson 3: Data types

You have learnt how to define a variable and print their values as well as display messages. But have you noticed something earlier while declaring Variables, when I was giving the value to the variables I used quotes sometimes and sometimes not. But what was the reason for that? When do we have to use quotes and when do we not? This can be clearly understandable while learning data types used in python.

Data Types can be defined as the classification of data items. It shows a kind of value a variable/object can perform.

There are many data types used in Python. For example, numeric data can have data type as an integer(int), floating point (float) and a complex number.
A Boolean data type is used when there are only 2 outcomes for a usage as True(T) or False(F) and also T can be considered as 1 and F as 0.
Normal uni-code characters can be used with data types as strings.

Numeric data types

Integers

Integers are referred to as **int** in Python. There is no limit to how long an int variable can be in python. Int variables are basically numbers without any decimals.

Example:

```
Marks = 74
Total_Marks = 100
```

Floating point

Floating point variables are represented by **float** class in Python. They are the number values which have decimals.

Example:

```
pi_value = 3.74
Speed = 147.5
```

Complex numbers

Complex numbers are represented by the **complex** class in python. Complex numbers consist of a real part and an imaginary part.

Example:

```
c = 1 + 4j
d = 66 - 64j
```

Strings

Strings can be used to store character data in Python. A single character is a string of length 1. Strings can be created using single, double and as well triple quotes.

Strings with single quotes

They are used to create & store single line characters.

Example:

```
first_name = 'Ricky'
last_name = 'Ponting'
```

Strings with double quotes

They are also used to store characters in a single line.

Example:

```
first_name = "Matthew"
last_name = "Hayden"
```

Strings with triple quotes

They are used to store characters from multiple lines.

Example:

```
multiple = ''' This is a multiple line string.
Opens with triple single quotes and
Ends with triple single quotes '''
```

Boolean

Boolean data are represented by **bool** class in Python. They are used when an object can only have value as True or False.

Example:

```
is_student_pass = True
is_student_fail = False
```

Note: The T and F in True and False should always be capital respectively. Otherwise, there will be an error as follows:

```
is_student_pass = true
is_student_fail = false
```

Output window:

```
NameError: name 'true' is not defined
```

You can see, we tried to declare True starting with small Case and eventually it throws an error.

Answer 2
Have you answered? Q.2? If not then do try.
Let's cross check the statement, first we declare a variable named age having value 20 then we use the print statement. Nothing difficult I guess. Let's crack the print statement, first object is ("Age =") it's in the quote so it will be displayed as it is then the second object is ("Age") this object is also inside the quotes so it will also be displayed as it is. After combining both the objects the output will look like as
Age = Age which means **option (d).** Were you correct this time or you have guessed the option (a)? I hope you don't need any more explanation why the answer is option (d).

So, now you have learned many data types. Let me show you a built-in function that can show what is the type of a variable.

type() is used to determine the data type of a particular variable.

Syntax:

type(variable_name)

Example:

```
roll_number = 47
name = 'Alex'
alive = True

print(type(roll_number))
print(type(name))
print (type(alive))
```

Output window

```
<class 'int'>
<class 'str'>
<class 'bool'>
```

As you can see that, we have declared variables **roll_number**, **name** and **alive** and then printed their data types. The output returned their datatypes as '**int**', '**str**', '**bool**' respectively.

It's question time

Q.3 What will be the output of the following statement:

```
my_age = 20
my_name = Michelle
print(my_age)
print(my_name)
```

(a)
```
Michelle
20
```

(b)
```
20
Michelle
```

(c)
```
my_age
my_name
```

(d)
```
NameError: name 'Michelle' is not defined
```

Lesson 4: Operators

You have seen how data types can be used and created in Python. Now let's see what kind of operations you could perform on these variables. Just Don't think operations are something related to doctors only. We use addition, subtraction and many more mathematical concepts in our day-to-day life, these are all considered as Operations.
There are many types of Operators that can be performed in Python.

Arithmetic operators

They are used to perform mathematical operations.

I am using two variables v1 and v2 to show you how arithmetic operators are performed in Python.

```
v1 = 8
v2 = 2
```

Operators	Operations	Syntax	Example	
			Code	Output Window
+	+ Operator is used to perform addition between the objects	sum = variable_1 + variable_2 …. +variable_n	`sum = v1 + v2` `print(sum)`	`10`
-	- Operator is used to perform subtraction between the objects	minus = v2 - v1	`minus = v2 - v1` `print(minus)`	`-6`
*	* Operator is used to perform multiplication between the objects	mult = v1 * v2 …. vn	`mul = v1 * v2` `print(mul)`	`16`
/	/ Operator is used to perform division between the objects.	divv = v1 / v2	`divv = v1 / v2` `print(divv)`	`4.0`
//	// Operator is used to perform floor division	di = v1//v2	`di = v1 // v2` `print(di)`	`4`

	between two objects.			
%	% Operator is used to get the modulus i.e remainder when dividing two objects	mo = v1 % v2	`mo = v1 % v2` `print(mo)`	`0`
**	** Operator is used when powering between two operators is performed.	Po = v1 ** v2	`Po = v1 ** v2` `print(Po)`	`64`

Answer 3
Let's see if you answered Q.3 correctly or not. I'll directly give you the answer this time, it's **option (d).** Let's decode why so. First, we have declared a variable named my_age which is a number. Then we declared a variable named my_name which is a string. But wait a minute, strings are declared using quotes. Could you see any quotes on that statement?
See you got your answer.

Q.4 What will be the output of the following statement:

```
a = 12
b = 2
print(a*b/a**b)
```

(a) `0.09777777777`

(b) `1.5555555555`

(c) `0.16666666666666666`

(d) `1`

Comparison Operators

They are used to obtain **True** or **False** values when comparing the values of variables.
They return either True or False depending on the comparison.
I am using two variables v1 and v2 to show you how comparison operators are performed in Python.

```
V1 = 17
V2 = 45
```

Operators	Operations	Syntax	Example	
			Code	Output Window
==	It is used to check that the objects are equal.	V1 == V2	`print(V1==V2)`	`False`
<	It is used to check that the left operand is less than the right operand.	V1 < V2	`print(V1<V2)`	`True`
<=	It is used to check that the left operand is less than or equal to the right operand.	V1 <= V2	`print(V1<=V2)`	`True`
>	It is used to check if the left operand is greater than the right operand.	V1 > V2	`print(V1>V2)`	`False`
>=	It is used to check that the left operand is greater than or equal to right operand	V1 >= V2	`print(V1>=V2)`	`False`
!=	It is used to check that both operands doesn't match.	V1 != V2	`print(V1!=V2)`	`True`

Logical Operators

We used these operators to perform logical AND, OR, NOT operations. They are used to combine conditional statements. They usually return **True** or **False**.
I am using two variables v1 and v2 to show you how comparison operators are performed in Python.

```
p = True
q = False
r = True
s = False
```

Operators	Operations	Syntax	Example	
			Code	Output Window

and	It will return true if all the conditions are true else it will return false	V1 and V2	`print(p and q)`
			`print(p and r)`
			`print(q and s)`
				`False`
				`True`
				`False`

or	It will return true if atleast one of the conditions are true,else it will return false.	V1 or V2	`print(p or q)`
			`print(p or r)`
			`print(q or s)`
				`True`
				`True`
				`False`

not	It will reverse the condition value. Suppose if the value is true then it will return false and if false then true.	not V1	`print(not p)` `print(not q)`	`False` `True`

Answer 4
Let's discuss the answer for Q.4, we have 2 variables a and b. Then the print function consists of an equation (a*b/ab), we know that *, /, ** is used for multiplication, division and exponential respectively. Just like BODMAS in mathematics, there exists operator precedence in Python. It follows as: parenthesis (), multiplication *, divide /, modulus %, floor division //, addition +, subtraction -, etc. There's more operator, but for now a programmer must know at least these precedence. So, comparing our equation with the precedence, the order of operations will be as follows: a**b(exponential) which will be 144 then a*b (multiplication) which will be 24 and then finally the value of {(a*b divide by a**b) division} which gives us our final answer as 0.166.**

Assignment operators

We use assignment operators to assign values to variables.

Operators	Operations	Syntax	Example	
			Code	Output Window
=	It is used to assign a value to the operand named on the left side.	A = B	`x = 12` `print(x)`	12

+=	It is used to add and assign value to the operand present on the left side.	A += B	`x = 12`
			`x += 5`
			`print(x)`
				17

-=	It is used to subtract value from the right side of the operator to the left side of the operator.	A -= B	`x = 12`
			`x -= 5`
			`print(x)`
				7

*=	It is used to multiply the right side operand to left side operand and then assign it to left operand.	A *= B	`x = 12`
			`x *= 5`
			`print(x)`
				60

/=	It is used to divide the right side operand to left side operand and then assign it to left operand.	A /= B	`x = 12`
			`x /= 5`
			`print(x)`
				2.4

%=	It is used to calculate the modulus (remainder) of the right side operand to left side operand and then assign it to left operand.	A %= B	`x = 12`
			`x %= 5`
			`print(x)`
				2

//=	It is used to divide (float) the right side operand to left side operand and then assign it to left operand.	A //= B	`x = 12`
			`x //= 5`
			`print(x)`
				2

**=	It is used to power the right operand with left operand and then assign it to left operand	A **= B	`x = 12`
			`x **= 5`
			`print(x)`
				`248832`

&=	It is used to logically AND the right operand with the left operand and then assign the value to left operand	A &= B	`x = 12`
			`x &= 5`
			`print(x)`
				`4`

|=	It is used to logically OR the right operand with the left operand and then assign the value to left operand	A |= B	`x = 12`
			`x |= 5`
			`print(x)`
				`13`

It's time for another question!

Q.5 What will be the output of the following statement:

```
a,b,c,x = 8,41,129,2
x *= ((b//a)+(c//b))
print(x)
```

(a) `SyntaxError: invalid syntax`

(b) `16`

(c) `3`

(d) `3.3414634146341466`

Lesson 5: Comments

Now, suppose that you have become a very popular programmer such that a company wants you to create a huge application. So, that application must have a large code as compared to the normal programs you use in our day-to-day life. Due to that some might get confused that which part of code is used for which purpose of the application. To solve this type of confusions, programmers use comments in between the program such that they get to remember the purpose of that particular code. They are generally used for the explanation of a particular code in a program.

Python Comments: Comments are used in Python for the better understanding of the code for the programmers. They are the lines which are not compiled by the compiler during the execution of he programmer.

There exists 3 ways to write comments in Python

Single Line Comments: To write a comment in just a single line in Python, we use hashtag (#) symbol. Anything written after # will be considered as a comment till the end of the line.

Multiline Comments: We can define multiline comments in Python in various ways. One way is to use # symbol at the beginning of each line and the other way is to use triple quotes (‘’’ or “””).

Example

```
#This is a single line comment

'''This is a multiline comment
Using triple single quotes
'''

"""
This is a multiline comment
Using triple double quotes
"""
```

Lesson 6: String Operations

So far so good. You understand the meaning of strings in Python. Let's see what kind of operations we can perform on these strings
Firstly, let us create a string.

```
string_1 = "Hii programmers"
```

So now we have successfully declared a string. Suppose we want to see how many characters are available in that string, what should we do? Should we count that one by one with our naked eye? Of Course not,while we are actually programmers.
Here we use a **function** which is available to us to use freely. It is a built-in function in Python. We can create our own function, which we will learn about more later.

len() : This function is used to return the number of characters present in a string.
Syntax

len(variable_name)

```
print(len(string_1))
```

Output window

```
15
```

Note: len() will count all the characters including blank spaces,underscores,etc.

If we want to check a particular letter in a string then we use,
count() : This function is used to return the number of occurrence(s) of a given character from a string.
Syntax

variable_name.count('character')

```
print(string_1.count('r'))
```

Output window

```
3
```

Answer 5
Let's Discuss the answer for Q.5 As we know that we can declare multiple variables in a single line separated by commas. Firstly, we have declared four variables a, b, c, x and their values as 8, 41, 129, 2 respectively. Then we tried to change the value of x as we have used the multiply and reassign (*=) assignment operator where we have equation like (b//a) + (c//b). So, The value for (b//a) would be 5 as floor division (//) is used and for (c//b) the value would be 3. Then these two values would be added which will give us 8. After that, previously the value of x was 2 so it will get multiplied with 8 and will give us the new value of x as 16. Pretty straight forward if you would have solved it calmly.

Now let's suppose we have declared a string, but we want to add more characters to that string, there we will use the concatenation operations on that string.
Simply we use + operator to add new string to the older one.

```
string_1 = "Hii programmers"
post = "!!!"
string_1 += post
print(string_1)
```

Output window

```
Hii programmers!!!
```

Note: Only string datatypes can be concatenated with the strings which means if you will try to concatenate a string with other data types it will raise and error.

To **lowercase**, **uppercase** or **capitalise** the characters of the string, we use the following functions:

lower() : It is used to convert all the characters of the string into lowercase.

Syntax:
variable_name.lower()

```
ai_name = "Hazel"
ai_name = ai_name.lower()
print(ai_name)
```

Output window

```
hazel
```

upper() : It is used to convert all the characters of the string into uppercase.

Syntax:
Variable_name.upper()

```
name = 'clark'
name = name.upper()
print(name)
```

Output window

```
CLARK
```

capitalize() : It is used to capitalise the first character of the string.

Syntax:
Variable_name.capitalize()

```
name = 'clark'
name = name.capitalize()
print(name)
```

Output window

```
Clark
```

Similar to capitalise(), we have,
title() : it is used to return the string in title format.

Syntax:
Variable_name.title()

```
name = 'nICHole'
print(name.title())
```

Output window

```
Nichole
```

We could also interchange all the cases of the characters using a function named,
swapcase() : It is used to invert all the cases of the characters in a string.

Syntax:
Variable_name.swapcase()

```
name = 'nICHole'
print(name.swapcase())
```

Output window

```
NichOLE
```

Now you have understood some of the functions related to strings.
Let's consider a real-life situation. Suppose you are designing a form where you are asking the user to enter their name and phone number for a survey.

To get the input from the user we use a function as,
input(): It is used to get the input from the user.
Syntax: input(message)
Here, message is optional

So, let's continue to take user name and phone number.
We will declare a variable named **user_name, user_ph_number** which will take users name and phone number respectively.

```
user_name = input ("Please Enter your name : ")
ph_number = input ("Please Enter your phone number : ")
```

This way you could take input from the user. But what if the user is some fancy person who likes to mess with the text by giving 'n' number of spaces in his/her name. To overcome this kind of situation we have some string functions:
lstrip(): It is used to remove all the spaces at the beginning of the string.

Syntax:
Variable_name.lstrip()

```
wrong_value = '       hahah'
print("Before L-Stripping wrong_value ",wrong_value)
print("after L-Stripping wrong_value",
      wrong_value.lstrip())
```

Output Window

```
Before L-Stripping wrong_value        hahah
after L-Stripping wrong_value hahah
```

rstrip(): it is used to remove all the spaces at the end of the string.

Syntax:
Variable_name.rstrip()

```
wrong_value = 'hahah         '
print(wrong_value,"is wrong_value before R-stripping")
print(wrong_value.rstrip(),"is wrong_value after R
    -stripping")
```

Output Window

```
hahah         is wrong_value before R-stripping
hahah is wrong_value after R-stripping
```

strip(): It is used to remove all the spaces from the beginning as well as the end of the string.

Syntax:
Variable_name.strip()

```
wrong_value = '     hahah    '
print(wrong_value,"is wrong_value before stripping")
print(wrong_value.strip(),"is wrong_value after stripping")
```

Output Window

```
hahah     is wrong_value before stripping
hahah is wrong_value after stripping
```

Also, if you want to not store some insensitive words from the user, you can replace it with some of your own words.

Suppose you do not want to have words such as "hell" in your input.
So, you will have to use function
replace() : It is used to replace the characters from the string.

Syntax:
variable_name.replace('character_to_replace',' character_to_be_replaced_by')

```
wrong_value = '     h a h a h    '
print(wrong_value.replace(' ',''))
# Here we have used replace to remove blank spaces
```

Output Window:

```
hahah
```

Also, to check if the user has actually entered a number in the phone number field, we have a function
isdigit(): It is used to check if the entered string contains only numbers. If yes, then it will return True or else will return False.
Likewise, if we want to check that if the user has entered only alphabets in the name input we have function,

Syntax:
Variable_name.isdigit()

isalpha(): It is used to check if the entered string contains only alphabetical characters.

Syntax:
Variable_name.isdigit()

```
my_name = 'Clara'
My_age = 15

print("Is My_name a digit :",my_name.isdigit())
print("Is My_name an alpha :",my_name.isalpha())
print("Is My_age a digit :",my_name.isdigit())
print("Is My_name an alpha :",my_name.isalpha())
```

Output Window:

```
Is My_name a digit : False
Is My_name an alpha : True
Is My_age a digit : False
Is My_name an alpha : True
```

There are many more Functions related to strings, but these are the ones which one must know.

It's time for another question!

Q.6 What will be the output of the following statement:

```
caller_name = 'stuart'
caller_number = 91174100811

full_details = caller_name.title() + caller_number
print(full_details)
```

(a)
```
stuart 91174100811
```

(b)
```
TypeError: can only concatenate str (not "int") to str
```

(c)
```
Stuart 91174100811
```

(d)
```
full_details
```

Lesson 7: Conditional statements

Now we have seen how we will know if a user entered proper data in our input field or not. So, when the user enters the wrong formatted input we have to prompt the user that they have not entered the proper data. This can be done by our latest topic: **Conditions.**
Conditional statements are used when we have to make decisions depending on certain conditions. We have to do tasks according to the conditions of satisfaction.
Most of you might have heard of the **"What..if.."** series by **Marvel**. Conditional statements are like that only, we have to perform if that happens then this will happen else that will happen. A lot of this and that, huh!

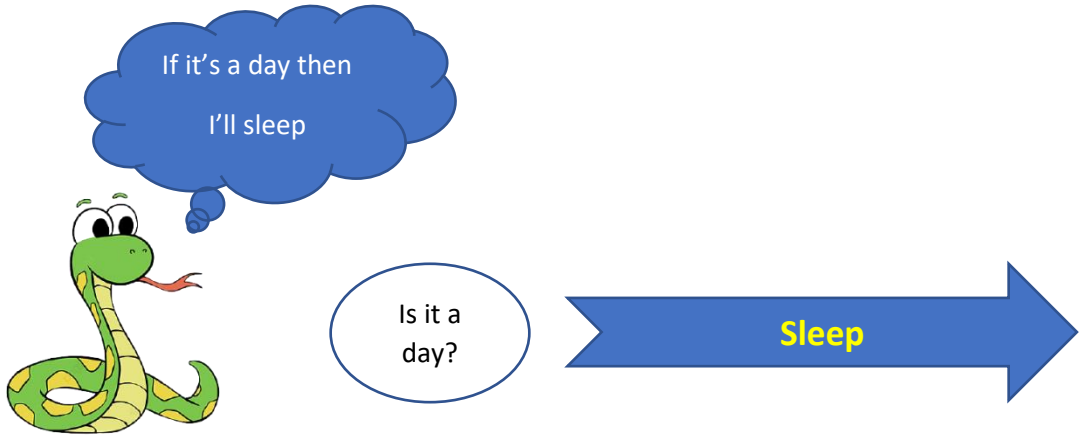
Firstly, we have a simple **if statement.**
Here we only have one if condition. Here a block of statement(s) is executed only if the condition specified is satisfied.

Syntax:
if condition:
    statement1
statement2
Suppose we have to check if a number if greater than 0 then we use

```
numb = 1
if numb > 0:
    print("Number greater than 0")
```

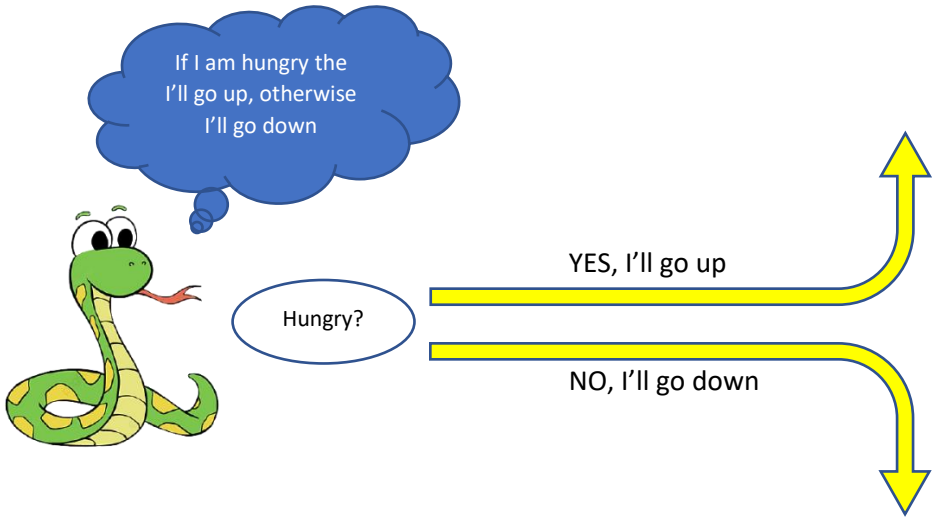
Output Window

```
Number greater than 0
```



if-else statement : it is used to check the condition and execute a set of statement(s) if the condition specified is satisfied or not.

Syntax
if condition:
    Statement1
else:
    Statement2
Statement 3



This kind of statement is used when if one condition doesn't get satisfied, then an else block statement will be executed.
Say if the number is not greater than 5 then we have to print it.

```
numb = 0
if numb > 5:
    print("Number greater than 5")
else:
    print("Number not greater than 5")
```

Output window:

```
Number not greater than 5
```

if-elif-else statement: It is used when we have to check for multiple conditions and if none matches we have an else block of statement to be executed.

Syntax

```
if condition1:
        statement1
elif condition2:
        statement2
else:
        statement3
statement4
```

Example: Suppose we want to check if a number is positive, negative or zero, then we will be required of **if_elif_else statement** as follows:

```
Number = 121
if Number > 0:
    print("Number is positive")
elif Number < 0:
    print ("Number is negative")
else:
    print("Number is not positive nor negative, means
        0")
```

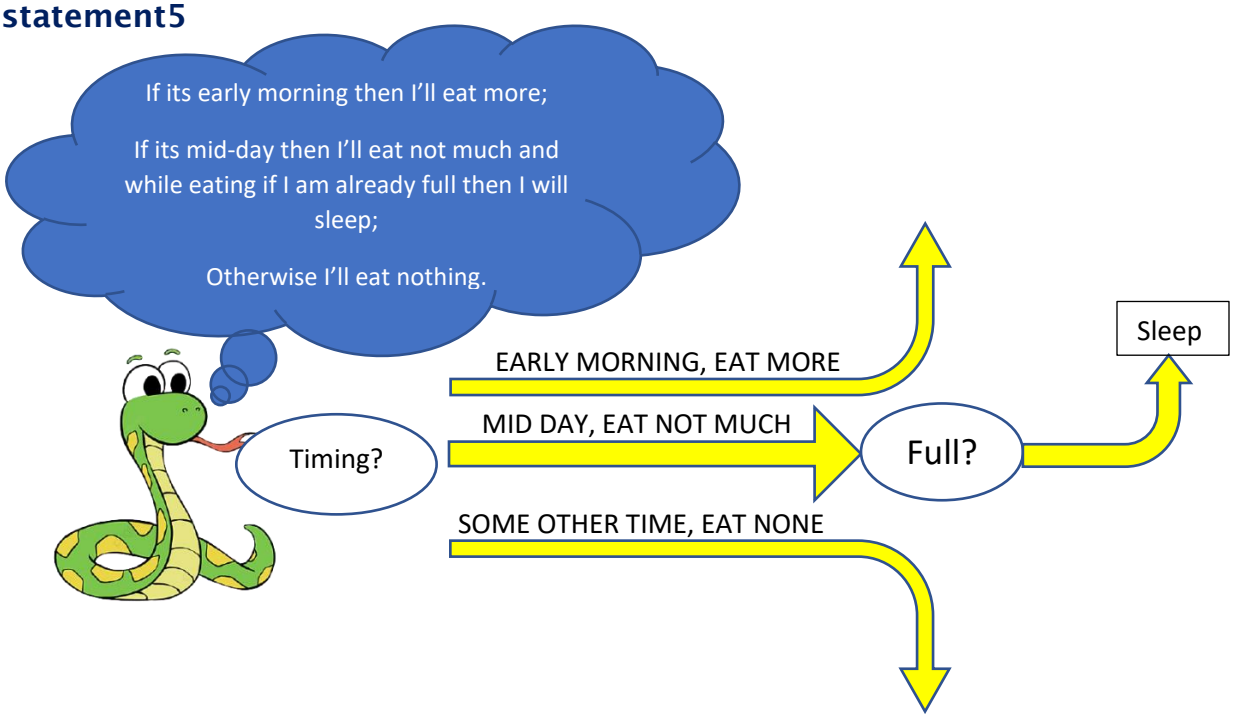
Output window:

```
Number is positive
```

Nested if: This statement is used when you have conditions within another condition.

Syntax

```
if condition1:
        statement1
        if condition2:
                statement2
elif condition3:
        statement3
else:
        statement4
statement5
```



Example: Suppose we want to declare a user as an adult or not depending on their age and gender. If the gender is not male or female then display "NA". Here we are assuming a male with age 20 or more and female with 18 or more is considered as an adult.

```
user_age = input("Enter your age : ")
user_gender = input("Enter your gender : ").title()
if user_gender == 'Male':
    if user_age >= '20':
        print("You are an adult")
    else:
        print("You are not an adult")
elif user_gender == 'Female':
    if user_age >= '18':
        print("You are an adult")
    else:
        print("You are not an adult")
else:
    print("NA")
```

Output window:

For a non-adult

```
Enter your age : 14
Enter your gender : female
You are not an adult
```

For an adult

```
Enter your age : 20
Enter your gender : male
You are an adult
```

Answer 6

Let's Discuss the answer for Q.6 Firstly, we declare two variables caller_name and caller_number with data types as string and int respectively. Then, we created a string variable full_details which will store both the data in a single variable and print it. The obvious answer will be option(b) as I have already mentioned that only strings can be concatenated with the strings. I hope you have answered correctly and not thought that answer would have been option(a).

Lesson 8: Iterative statements

You all might have played games. If not, you might have a basic knowledge of some games that if you play a game, you win or lose. I know people say you do not lose in a game, You either win or Learn!! Haha, these things only feel good to be motivated, in the real world we all know how it feels to lose. We know the feeling of winning too. But let's stick to the programming world, so the game developers have to write codes such that while the user is playing the game and they have not finished the game, the game shall continue to run. Have you seen anyone playing a game and suddenly the game stopped? I guess you might have seen this scenario, but let me make a thing clear this scenario happens because of the game not supporting the hardware of the device. There is nothing wrong with the game, it's the device issue. You might have played or are still playing action games such as Counter Strike, Pubg , Cod, Bgmi, etc. When you are playing a classic match, does the match complete without someone getting a win?
These kinds of operations are usually done by using iterative statements, such that the game continues to run until the player has won or lost.

Q.7 What will be the output of the following statement:

```
my_age,your_age = 7,6
my_exp,your_exp = 0,0
if my_age < your_age:
    print("you are older than me")
    if my_exp <= your_exp:
        my_exp += 1
else:
    print("I am older than you")
    if my_exp >= your_exp:
        your_exp += 1

print('My exp is:',my_exp,'and yours is:',your_exp)
```

(a)
```
I am older than you
My exp is: 1 and yours is: 0
```

(b)
```
I am older than you
My exp is: 0 and yours is: 1
```

(c)
```
You are older than me
My exp is: 0 and yours is: 1
```

(d)
```
you are older than me
My exp is: 1 and yours is: 0
```

Iterative statements: In python we use loops to run a set of code again and again until the condition satisfies.

while loops: while loops are used when we want to execute a block of statements until a specific condition is satisfied. When the condition doesn't satisfy, then the loop will be broken and it will come out of the loop, which means the code outside the loop will start running.

Syntax:

while condition:

    statements

Example: Obviously, I am not going to give you an example of how battleground games logic is written.
Let's write a code for printing the statements 3 times.
We have written this code such that it will continue to run until the value of variable st_count becomes 3. The loop will increment its value by 1 each time after printing the statement.

```
st_count = 0
while st_count < 3:
    print ("Statement printed")
    st_count += 1
```

Output window:

```
Statement printed
Statement printed
Statement printed
```

for loops: for loops are generally used as traversals. They are used to check in particular strings, lists, dictionaries, etc until the condition satisfies.
There are many ways in which we can define a for loop.

Syntax:

for iterator in traversal:

    statements

here traversal can be anything like a string, list, etc or a range of functions.

Example: Writing a program such that it will return the characters of string one by one.

```
string_value = "Python"
for I in string_value:
    print(I)
```

Output window:

```
P
y
t
h
o
n
```

Answer 7
The answer for Q.7 will be option(b). As the first condition in the if statement is to check whether my age(7) is less than your age(6) which doesn't satisfies so it will directly goes to the else block and prints corresponding statement and then it checks my_exp(0) is greater than or equal to your_exp(0), as it gets satisfies the value of your_exp gets incremented by 1. Finally, the exp of both is printed.

It's time for another question!

Q.8 What will be the output of the following statement:

```
numb = 7
tep = 1
while numb > 5:
    tep += 1
    numb -= 1
    print(tep*numb)
```

(a)
```
7
5
```

(b)
```
12
9
```

(c)
```
12
15
```

(d)
```
12
LoopError : Cannot loop much
```

Lesson 9: Functions

You all have seen the string functions such as upper(), lower() and many. Now, let us create our own functions.
Functions: They are a Block of statement used to perform logic to get the desirable results. It's a block of code which can only be run when a function is called.

In python, we use **def** keyword followed by function_name and then by parenthesis()
Syntax:

```
def function_name():
        Statement1
        Statement2
```

In the parenthesis, we could pass values which are generally termed as "**arguments**" in Python.
Arguments are passed depending on the function.

Example: Let's create a function that will return a statement.

```
def show():
    print("This is a function")
```

Now we have created the function, but how do we perform it? To run a particular function we have to call it.

Function calling: Inorder to run a particular function, we have to call it using its name followed by parenthesis.
So, to run the above function
We will write

```
show()
```

Output window:

```
This is a function
```

As we discussed earlier, we can pass arguments in the parenthesis of the functions. Let us take an example of such a case as if we want to return the square of a number.

```
def squaring(n):
    Sq = n*n
    print(Sq)
```

Here we have written a code for squaring a number where we have used a variable named n to get the argument.

To call the function we will write,

```
squaring(4)
```

Output window:

```
16
```

You see we have passed **4** in parenthesis. Here **4** is an **argument** to the function named squaring which will return 4*4 which is 16.
Similarly, we can write a program for returning a cube of a number.

```
def cubing(n):
    cu = n*n*n
    print(cu)

cubing(7)
```

Output window:

```
343
```

Answer 8
Comparing to the Other questions, the answer for Q.8 is quite easy. We simply declared two variables and then created a loop such that the code will until the variable numb(7) is greater 5. So, first time the condition is satisfied as 7 > 5 and then the value of tep is incremented by 1 and the value of numb is decremented by 1 which leads to value as 2 and 6 respectively and then they are multiplied and its value is printed. As the block is over, the condition is checked again. Now, numb(6) > 5, it still gets satisfied so now the value of tep and numb will be 3 and 5 respectively. So, the multiplication of tep(3) and numb(5) will result in 15. After this, again the condition is checked, numb(5) > 5 now the condition does not get satisfied and the loop is being breaked resulting in exit of code. So, the final answer will be 12 and 15.

Q.9 What will be the output of the following statement:

```
Name = "Alicia"
Age = 24
Profession = "Bio technologist"

def my_data():
    print("My name is ",Name)
    print("My age is ",Age)
    print("My profession is ", Profession)
```

(a)
```
My name is Alicia
My age is 24
My profession is Bio technologist
```

(b)
```
My name is Alicia My age is 24 My profession is Bio
    technologist
```

(c)
```
Alicia
24
Bio technologist
```

(d)
```
>
```

Do you remember the program we were writing for the users name and age for a survey? There we see how the user has entered the name and age in proper format. Then we also displayed the message to the user using certain conditions that they have not entered the data correctly. But let's do something such that if the user entered the data in the wrong format, then we will prompt the user to re-enter the data again. Sounds interesting?
Firstly, we will write a function which will take the input from the user.

```
def user_info():
    user_name = input("Please Enter your name : ").title()
    user_ph_number = input("Please Enter your phone number : ")
```

Now let's write a function to verify the above details by creating a function, if the data is not entered properly, then it will redirect the user to enter the details again.

```
def is_data_correct(user_name,user_ph_number):
    if (user_name.isalpha()) and (user_ph_number.isnumeric()) == True:
        print("Data entered properly")
    else:
        print("Data entered is not correct")
        user_info()
```

Complete program

```
def user_info():
    user_name = input("Please Enter your name : ").title()
    user_ph_number = input("Please Enter your phone number : ")
    is_data_correct(user_name,user_ph_number)

def is_data_correct(user_name,user_ph_number):
    if (user_name.isalpha()) and (user_ph_number.isnumeric()) == True:
        print("Data entered properly")
    else:
        print("Data entered is not correct")
        user_info()

user_info()
```

Output Window

```
Please Enter your name : ed
Please Enter your phone number : sheeran
Data entered is not correct
Please Enter your name : ed
Please Enter your phone number : 8114565841
Data entered properly
```

Detailed explanation

First, we created a function named user_info() where we created two variables which take the input from the user which stores the user's name and phone number respectively.
Then, we created a function which checks if the name value only contains alphabetical characters and phone number only contains numeric values, we combine this both conditions using logical (**and) operator** which expects both the conditions to be satisfied then only it will print that the data is entered properly, otherwise it will call the user_info() to get the input from the user again.

These all can be done once we have called the function user_info() as we have done at the end of the program. Also, we have used parameters(arguments) for the is_data_correct() function so that the vale of user name and user phone gets passed to the function.

Now, we have understood that a function can call other functions. But also, a function can call itself within its block of statement, this type of functions are called **Recursive functions**.

Recursion: Recursion means getting repeated again and again.

Example; Let us consider an example of a Recursive function which will return a factorial of a number.
Spoiler alert: Factorial can be defined as, in mathematics, the product of all positive integers less than or equal to a given positive integer and denoted by that integer and an exclamation point.
Hence, **Factorial of 3 is (3! = 3 * 2 *1 = 6)**

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))

num = 3
print("The factorial of ",num,"is", factorial(num))
```

We define a function where if the input is 1 then the Factorial is returned 1 directly, otherwise it will multiply itself with one less and again goes back to the top of the function, this goes on until the value of x is 1.
Our recursion ends when the number reduces to 1. This is called the base condition.

Note: Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

Answer 9
The Ones who have answered Q.9 as Option(a), Let me tell you that you have answered Incorrectly. Let's see how. We have declared 3 variables Name, Age and Profession, Then, we have created a function my_data that will be printing the respected data of Name, Age and Profession. But, have we called the function in the program so that the printing of the data will be done? As discussed earlier the function will only be executed once its being called. So, the answer will be option(d)

It's time for another question!

Q.10 What will be the output of the following statement, input is given as 7 when asked:

```
def recur_(n):
   if n <= 1:
      return n
   else:
      return n + recur_(n-1)

user_value = int(input("Enter a positive number: "))
if user_value < 0:
   print("Only Enter a positive number")
else:
   print("The final value is",recur_(user_value))
```

(a)
```
Enter a positive number: 7
The final value is 5040
```

(b)
```
Enter a positive number: 7
The final value is,recur_(user_value)
```

(c)
```
Enter a positive number: 7
The final value is 28
```

(d)
```
Enter a positive number: 7
Only Enter a positive number
```

Lesson 10: Advance Data types

Lists

Python lists are one of the most versatile data types that allow us to work with multiple elements at once. This means that we can store multiple types of data in a single list, like we can store string as well as numbers in a single list.
A list can have any number of items.
A list is created by placing elements inside square brackets [], separated by commas.

Syntax
list_name = [value1,value2,.....valuen]

Example: creating a list which contains the name of persons.

```
person_list = ["Ricky","Kacey","Steve"]
```

A list with different data types

```
new_list = ["Me",18]
```

We can also create a list with another list.

```
person_list = ["Ricky","Kacey","Steve"]
new_list = ["Me",18]
combined = [ new_list , person_list , [3,4] ]
```

Accessing the list

We can access the elements of the list with 2 indexing methods:
Positive and **Negative.**

Positive indexing starts with **0** and not with 1 in python, which means the indexing sequence **is 0, 1, 2, … n.**

Negative indexing starts with **-1** in python, when means indexing sequence is **-1, -2, -3, …. n**
For example, let's create a list containing names of dogs.

```
names = ["Marky","hella","susey","kalu"]
```

To access the first element with positive indexing, we will write

```
print(names[0])
```

Output window:
```
Marky
```

To access the last element with negative indexing, we will write
```
print(names[-1])
```

Output window:
```
kalu
```

Likewise, we can return any elements from the list.

List slicing

As we can access single elements, we can also access a range of elements from a list which can be termed as Slicing.
The slicing operator used in python is:

Slicing Queries	Syntax/Example	Output window
To print all the elements from the list we do not specify any index.	`print(names[:])`	`['Marky', 'hella', 'susey', 'kalu']`
To print the elements from an index to the last element of the list	`print(names[2:])`	`['susey', 'kalu']`
To print the elements from the beginning to certain index.	`print(names[:2])`	`['Marky', 'hella']`
To print certain range of elements.	`print(names[2:4])`	`['susey', 'kalu']`

Note: When we slice lists, the start index is included but the end index is excluded. For example, names[2: 4] returns a list with elements at index 2 and 3, but not 4.

List Updation

Lists are mutable, which means we can add and remove elements from the lists without throwing any errors.
We can add, remove and update an element of a list.

Adding

Let's suppose we have created a list of my favourite ice cream flavours.
```
my_list = ["pista","chocolate chip","strawberry"]
```

But now I remember that I also love Vanilla, so I have to add it into the list.
append(): It is a method used to add an element to the end of the list.

So, we use this method to add vanilla to my_list as

```
my_list.append("Vanilla")
```

Let's check if we have successfully added Vanilla to the list or not

```
print(my_list)
```

Output Window:

```
['pista', 'chocolate chip', 'strawberry', 'Vanilla']
```

Again, you see here I could only add a single element with this method. To add multiple elements in the list we use extend().

extend(): It is a method used to add multiple elements to a list.
Let's add two more flavors.
Actually, let's just create another list of new flavors. The idea behind this is to show that we can add a list to another list.

```
new_list = ["custard","mint chocolate"]
```

Let's add the new flavors to the original list and see if it's being added or not.

```
my_list.extend(new_list)
print(my_list)
```

Output window:

```
['pista', 'chocolate chip', 'strawberry', 'Vanilla', 'custard',
    'mint chocolate']
```

Now Suppose if malai flavour is the one which I like the most such that it should appear at the beginning of my list.

insert(): It is used to add an element at a desired index of the list.

```
my_list.insert(0,"Malai")
print(my_list)
```

Output window:

```
['Malai', 'pista', 'chocolate chip', 'strawberry', 'Vanilla',
    'custard', 'mint chocolate']
```

Updating

Now, let's imagine that I don't want to put Mint chocolate in my list, I want butterscotch in that place. So, we have to see the index place of mint chocolate in the list

```
print(my_list.index("mint chocolate"))
```

Output window:

```
6
```

This way we can see the index of any element from a list

To update, simply we could do it like this and then print it,

```
my_list[6] = "Butterscotch"
print(my_list)
```

Output window:

```
['Malai', 'pista', 'chocolate chip', 'strawberry', 'Vanilla',
    'custard', 'Butterscotch']
```

Deleting

Now I realize that I actually don't even like much of Vanilla flavour.

remove(): It is used to delete an element from the list.

So, to remove vanilla from the list,

```
my_list.remove("Vanilla")
print(my_list)
```

Output window:

```
['Malai', 'pista', 'chocolate chip', 'strawberry', 'custard',
    'Butterscotch']
```

del[a:b] : It is used to delete elements within a range in a list.

```
del my_list[1:3]
print(my_list)
```

Output window:

```
['Malai', 'strawberry', 'custard', 'Butterscotch']
```

So that we have understood the list, now we want to remove all the elements of the list, we use

clear(): It Is used to clear all the elements of the list.

```
my_list.clear()
print(my_list)
```

Output Window: `[]`

As now we don't need our list, we can delete the list

```
del my_list
print(my_list)
```

After deleting the list, we tried to print the list as it will cause an error.

```
NameError: name 'my_list' is not defined
```

Answer 10
The answer for Q.10 is based on the principle of Recursion. Here, we assume that when asked the user inputs 7, This number will go on to the function again and again until the number 7 is less than equals to 1. While in the function each time the number is passed, it will be getting added to itself which means first 7 is passed then 6 where the returning value is 7+6 which is 13. Then, 5 is passed which will return into 13+5=18, this will go on until the limit of 1 is +reached. So, the total answer would look like as 7+6+5+4+3+2+1 which will be equal to 28. So, the answer is option(a).

It's time for another question!

Q.11 What will be the output of the following statement?

```
eating_list = ['noodles','frankie','toast','paneer']
print(eating_list[1:3])
eating_list.insert(4,'wafer')
print (eating_list[2:4])
```

(a)
```
['noodles','frankie','toast']
['frankie','toast','paneer']
```

(b)
```
['noodles','frankie']
['frankie','toast']
```

(c)
```
['frankie', 'toast']
['toast', 'paneer']
```

(d)
```
['frankie','toast','paneer']
['toast', 'paneer', 'wafer']
```

Tuples

Tuples are created and used to store multiple items in a single variable.
They are **immutable**, which means once they are created, we cannot add, remove or update Items on that tuple.
Similar to the lists, tuples are also indexed beginning with 0.

Creating a tuple

Syntax:

tuple_name = (item1, item2, ... item_n)

```
Names = ("Ray","Joy","Sia")
print(Names)
```

This tuple contains an item at 0th index as Ray, Joy at 1st index and Sia at 2nd index.Similar to Lists, we can access the elements of the tuples

```
print(Names[1])
```

Output window

```
Joy
```

Answer 11
In Q.11 we have created a list containing some food items and then prints the range of elements from the list. Then, we added another food item to the list at the index of 4. As we know the indexing starts from 0 in Python Lists and while slicing the range of elements the starting index number is included and the ending index is not included. In our case, for the first print statement the range is 1:3 where 1st (frankie) and 2nd (toast) element will be printed and in the second print statement, the range is 2:4 where 2nd (toast) and 4th (paneer) element will be printed. Thus, the answer will be option(c).

It's time for another question!

Q.12 What will be the output of the following statement?

```
my_tuple = (1,2,3)
your_tuple = (4,5,6)
print(my_tuple[1],"&",your_tuple[2])
```

(a)
```
1 & 5
```
(b)
```
2 & 6
```
(c)
```
TupleError: Cannot Print tuple Value
```
(d)
```
1,2,3 & 4,5,6
```

Dictionary

So far you have seen many data types which could store string, number data as well as multiple data's in a single variable.
But can you declare your whole bio data in a single variable like suppose your **name** is **Kelly**, **age** is **16, lives** in **California**, **gender** is **female,** and much more like this information. If you say yes, we could declare using a tuple or list like this

```
my_data = ("Kelly",16,"California","Female")
my_data = ["Kelly",16,"California","Female"]
```

But, where does it say that your name is Kelly and age is 16, the one who doesn't know you at all, or says doesn't understand how tuples or list work cannot determine what's this, California can also be your name, 16 can be your waist size, etc.
In order to achieve this, python has a data type called dictionary.

Dictionary: It is used to store a key: value pair. Unlike other data types, a dictionary can store values as key: value pair.

Creating a dictionary

Syntax
dictionary_name = {key1:value1,key2:value2,... key_n:value_n}

Creating a dictionary for a student

```
student = {'name': "Tess", 'age': 14, 'studying_in': 9, 'gender': "female"}
```

Here keys are name, age, studying_in and gender and values are Tess,14,9, female respectively.

Dictionary updation

We can add or remove and update values of the dictionary.

Adding

After creating the student dictionary, we see that we have missing data for a student. We have to add a roll number key and it's value.
To add roll_num and it's value,

```
student['roll_no'] = 17
print(student)
```

Output Window:

```
{'name': 'Tess', 'age': 14, 'studying_in': 9,
    'gender': 'female', 'roll_no': 17}
```

Updating

We now want to update the age of student, to do that,

```
student['age'] = 13
print(student)
```

Output Window:

```
{'name': 'Tess', 'age': 13, 'studying_in': 9,
    'gender': 'female', 'roll_no': 17}
```

Note: While updating a key's value, that key should exist in the dictionary, if not then it will be created.

Answer 12
The answer of Q.12 is quite easiest of all the questions. We created 2 tuples and then tried to print the value of certain elements of that tuple. First print object contains index 1 of my_tuple whose value is 2 and then a separator is used the the value of index 2 of your_tuple is printed whose value is 6. Hence, The answer is option(b)

It's time for another question!

Q.13 What will be the output of the following statement?

```
my_dict = {'name': "Hazel", 'age': 17, 'gender': "Female" }
my_dict['fav_color'] = "black"
my_dict['age'] = 19
print(my_dict)
```

(a)
```
{'name' : 'Hazel' , 'age' = 17 , 'gender' = Female}
```

(b)
```
{'name' : 'Hazel' , 'age' : 17 , 'age' : 19, 'gender'
    : Female}
```

(c)
```
{'name' : 'Hazel' , 'age' : 17 , 'age' : 19, 'gender'
    : 'Female','fav_color': 'black'}
```

(d)
```
{'name': 'Hazel', 'age': 19, 'gender': 'Female',
   'fav_color': 'black'}
```

Accessing

keys():It is used to get all the keys stored in the dictionary, we use

```
print(student.keys())
```

Output Window:

```
dict_keys(['name', 'studying_in', 'gender',
    'roll_no'])
```

values():It is used to get all the keys stored in the dictionary, we use

```
print(student.values())
```

Output Window:

```
dict_values(['Tess', 9, 'female', 17])
```

To access the value of particular key, we use

```
print(student['name'])
```

Output Window:

```
Tess
```

Deleting

Suppose if we do not want the age column, to delete age column

```
del student['age']
print(student)
```

Output window:

```
{'name': 'Tess', 'studying_in': 9, 'gender': 'female', 'roll_no': 17}
```

If we want to delete whole dictionary,

```
del student
print(student)
```

Output window:

```
NameError: name 'student' is not defined
```

Answer 13
The answer of Q.13 will be option(b) let's see how. First, we created a dictionary with 3 keys and values pairs which are name: Hazel, age: 17, gender: Female. Then we added fav_color: black. Then, we tried to add age, As age is already present as a key in the dictionary it will not be added to the dictionary instead it's value will get updated now as 19. Finally, the print statement will display the 4 key value pairs as name: Hazel, age: 19, gender: Female, fav_color: black.

Let's Conclude this part of our journey with one final questions. If you would able to solve the program, you can e-mail your answer with a simple program execution. My email address is rahulroy74810@gmail.com .

Write a python Program which will accept some inputs from the user. The inputs to be taken are User's First Name, User's Surname, User's Age, User's height and User's weight. Make sure that user can only enter string data in name fields and int data in age, weight and height. If the data entered is in wrong format, prompt the user about it and tell them to enter the data properly. If the data is correct, then calculate users HAW Ratio which will be {HAW=age+(weight/height)}. After Calculating HAW Ratio, print the user category as follows:
0 – 7.4: Underweight
7.41 – 13.50: Lightweight
13.51 – 21.60: Obese
21.61+: Heavyweight
Finally, show the data to user in following format:
[Your name is {User's Surname + FirstName in capital letters}
Your Age is {age} years old
You belong to {HAW RATIO category} Category
Thank You For Using the platform!]

Hint to solve above problems is to use Recursive Functions, string functions to convert case, Conditional Statements.

NOTE: I hope you learned a lot! Make sure to practice basic programming because there is huge difference is just reading and understanding.

So, Keep Practicing. Don't Just have a good life, Have a great Life!