Machine Learning Diploma

Python1: Setup Environment & Python Basics

AMIT

Agenda

- ➔ Intro to python
- ➔ Anaconda and virtual environments
- ➔ Jupyter
- ➔ Python basics syntax, comments, variables
- ➔ data types
- ➔ Print statement
- ➔ User inputs
- ➔ Strings, String methods

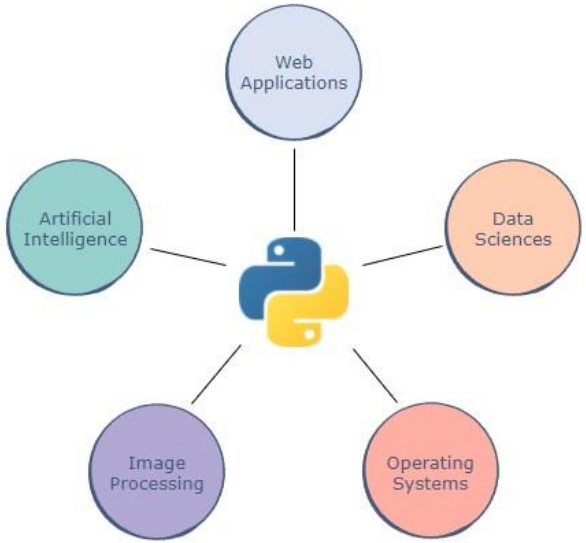
AMIT

1. Intro to python

AMIT

What is Python ?

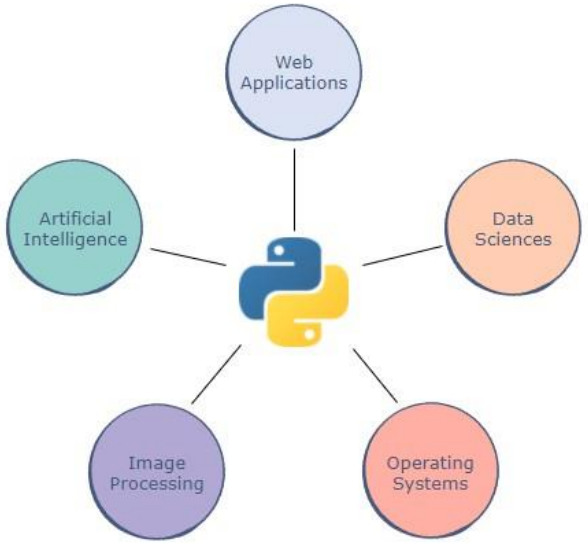
- ➔ Python is an interpreted, object-oriented, high-level and one of the most popular general-purpose programming languages in modern times.
- ➔ The term “general-purpose” simply means that Python can be used for a variety of applications and does not focus on any one aspect of programming.



AMIT

What is Python ?

- ➔ Python is an interpreted language because, during execution, each line is interpreted to the machine language on-the-go.
- ➔ Python is a very efficient language which is used in almost every sphere of modern computing



AMIT

INTERPRETED VS. COMPILED

Compiled Language	Interpreted language
Needs an initial build step to convert code to machine code that CPU understands	Doesn't need a build step but it is run line by line and interpreted on the go.
Faster in execution.	Slower in execution.
More control over the hardware as memory management.	Less control over the hardware.
Examples: C++, C, C#	Examples: Python, JavaScript, Perl

AMIT

PYTHON 2 VS PYTHON 3

Python2	Python3
More difficult in syntax	Simpler in syntax
Global variables can be changed	Global Variables can't be changed
Parenthesis weren't used in prints and exception	Parenthesis are used in prints and exception
Latest version is 2.7 in 2010	Latest version is 3.10.1 in jan 2022

AMIT

Why python for Machine Learning ?

➔ Python is easy to understand you can read it for yourself. Its readability, non-complexity, and ability for fast prototyping make it a popular language among developers and programmers around the world.

➔ Python allows easy and powerful implementation With other programming languages, coding beginners or students need to familiarize themselves with the language first before being able to use it for ML or AI. This is not the case with Python. Even if you only have basic knowledge of the Python language, you can already use if for Machine Learning because of the huge amount of libraries, resources, and tools available for you.

Why python for Machine Learning ?

➔ Python has a great library ecosystem Having access to various libraries allows developers to perform complex tasks without the need to rewrite many code lines. Since machine learning heavily relies on mathematical optimization, probability and statistics, Python libraries help data scientists perform various studies easily.

➔ Huge community Python has a strong group of supporters, and it’s worth knowing that if you experience a problem, there is someone out there who’ll be able to offer you a helping hand.

Python installation (Windows OS)

➔ Steps:

- Go to www.python.org on your browser.
- Navigate to the “downloads” tab and click python for (Windows) OS .
- Click on “Latest Python 3 Release - Python 3.x.x”.
- Choose “Windows installer (64-bit)” at the end of the webpage
- Navigate to the downloaded files and open the installer
- It’s a must to check the red box to add the python Path to the Environment variables of your system.
- In the “Advanced Options” section, check “Install for all users” option for multi users' access.

AMIT

2. Anaconda

What is Anaconda ?

- ➔ Anaconda is a free and open-source distribution of the programming languages Python and R that comes with the Python interpreter and various packages related to machine learning and data science.
- ➔ Basically, the idea behind Anaconda is to make it easy for people interested in those fields to install all (or most) of the packages needed with a single installation.
- ➔ As it is managed by An open-source package and environment management system called Conda, which makes it easy to install/update packages and create/load environments.
- ➔ Anaconda also comes with several other software additions like the Anaconda Navigator, a graphical user interface.

WHICH TO USE? ANACONDA OR MINICONDA?

Anaconda	Miniconda
Needs more disk free space (About 3GB)	Needs very minimal disk space.
Comes equipped with over 250 packages.	Comes with no pre-installed packages.
Has an interface called Anaconda Navigator as well as the command prompt.	Doesn't have a GUI interface but only the command prompt.

AMIT

Installing Anaconda

➔ Steps:

- Go to www.anaconda.com on your browser.
- Navigate to the “Products” tab and click “individual edition”.
- Click on “download” to download the windows installer.
- Navigate to the downloaded files open the installer.

➔ Additional Notes:

- Make sure the you have sufficient free disk space on the drive you are installing in.
- Check add Anaconda in my PATH environment variables in advanced options section.

AMIT

Virtual Environments

- ➔ Python packages, don’t always play nice with each other. Sometimes, you need different versions of things for particular projects. Python virtual environments feature, aka venv, was developed to offset this problem.
- ➔ A virtual environment is a Python environment such that the Python interpreter, libraries and scripts installed into it are isolated from those installed in other virtual environments. part of your operating system.
- ➔ Anaconda makes it possible to create a project-specific isolated virtual environment.

Anaconda Important Commands

Open the anaconda prompt/terminal:

- ➔ Create a new environment: $ conda create --name venv_name
- ➔ Activate the environment in conda: $ conda activate venv_name
- ➔ Install a new package: $ conda install package_name
- ➔ Check for installed packages: $ conda list
- ➔ Check for created environments: $ conda env list

Note: When creating a new virtual environment, it won't come packaged with the pre-installed packages in the root virtual environment.

➔ For more commands visit this link

AMIT

3. Jupyter Notebooks

AMIT

JUPYTER NOTEBOOK

- ➔ The Jupyter Notebook is a free, open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and as an educational tool .
- ➔ Jupyter notebooks are now the base of important platforms as Google Colab it is a Jupyter notebook that runs on Google servers with a free GPU and integrated with important libraries as pytorch, tensorflow, keras and openCV. and you don’t need to install anything. Moreover, the notebooks are saved to your Google Drive account.

AMIT

JUPYTER NOTEBOOK

- ➔ To activate your environment: $ conda activate venv_name
- ➔ To install Jupyter Notebook: $ conda install jupyter
- ➔ Jupyter Notebook, Type the following: $ jupyter notebook or $ jupyter-notebook

You will need to manually install Jupyter if you didn't install it when creating the virtual environment.

AMIT

Markdown cells

Jupyter Notebooks are constructed based on cells.
There are two types of cells: Code cells, Markdown cells.

Code cells: For executing code.
Markdown cells: For documenting notes and guides for the codes.

For example, you can specify a title for the project at the very top of the notebook by making a markdown cell and write > # Project Name
For level 2 titles, use ##. For level 3 titles, use ### and so on.
You can view this link for more syntax ideas to make your notebook more readable.

Markdown cells

- ➔ To add a cell above the current cell press a
- ➔ To add a cell below the current cell press b
- ➔ To delete a cell press d twice
- ➔ To convert a cell to markdown press m
- ➔ To convert a cell to code cell press c
- ➔ To run a cell: CTRL+Enter: run the current cell.
  SHIFT+Enter: run the current cell and move forward.
  ALT+Enter: run the current cell and insert a cell after.
- ➔ An asterisk (*) will appear while the cell is running. Once finished, The executed cell will have a number indicating its current order among all run cells.

AMIT

JUPYTER NOTEBOOK TIPS

Cell modes: Command mode (blue cell) and Edit mode (green cell)

To activate edit mode just click inside text box or press Enter.
To activate command mode just click outside the text box or press ESC.

Execution: To run a cell, we do one of the following:

CTRL+Enter: run the current cell.
SHIFT+Enter: run the current cell and move forward to next one.
ALT+Enter: run the current cell and insert a cell after.

An asterisk (*) will appear while the cell is running. Once finished, The executed cell will have a number indicating its current order among all run cells.

AMIT

JUPYTER NOTEBOOK TIPS

Adding and removing cells: Make sure you are in command mode.

To add a cell above the current one, press “a” once. To add it below, press “b” once. To delete a cell, press “d” twice.

Restart the kernel: Make sure you are in command mode. To

restart the kernel, press “0” twice.

You can copy and paste cells or even merge cells normally. Note:

The changes in the notebook is saved automatically.

AMIT

JUPYTER NOTEBOOK TIPS

Markdown:

You can add heading and subheadings in your notebook for organizing it by changing the cell type from code to markdown from the dropdown list.

To appear as a heading, just add # followed by space before the title.

One # means heading and by adding more # you will get a smaller font heading.

By changing the cell mode to command mode, you can easily press any key from 1 to 5 to control the heading font or level.

Don’t forget to press shift enter after adding the title to appear as a heading.

For more jupyter notebook tips:
https://www.edureka.co/blog/cheatsheets/jupyter-notebook-cheat-sheet

AMIT

4. Python Basics

Basics:

➔ Comments:

- Single comment: #
- Multiline comment/docstrings: between three double/single quotes “”” “””. Or you can just write # multiple times.
- You can use Ctrl+ “/” to automatically comment a whole section.

```
""" Docstrings are pretty cool
for writing longer comments
or notes about the code"""

```

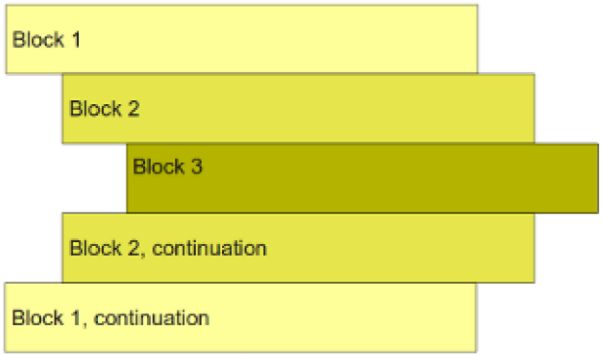
```
print(50)  # This line prints 50
print("Hello World")  # This line prints Hello World

# This is just a comment hanging out on its own!

# For multi-line comments, we must
# add the hashtag symbol
# each time

```


AMIT

Basics:

➔ Indentation:

- Indentation is Important in Python. It Illustrate how code blocks are formatted.




```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
  print ("False")
```

error

Basics:

➔ Print Statements:

- Since Python is one of the most readable languages out there, we can print data on the terminal by simply using the print statement.
- The text Hello World is bounded by quotation marks because it is a string or a group of characters
- Next, we'll print a few numbers. Each call to print moves the output to a new line

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

```
Hello World
```

```
print(50)
print(1000)
print(3.142)
```

```
50
1000
3.142
```


AMIT

Basics:

➔ Print Statements:

- We can even print multiple things in a single print command; we just have to separate them using commas
- By default, each print statement prints text in a new line. If we want multiple print statements to print in the same line, we can use the following code
- The value of end is appended to the output and the next print will continue from here

```
print(50, 1000, 3.142, "Hello World")
```

```
50 1000 3.142 Hello World
```

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


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

```
HelloWorld
Hello World
```

Quiz:

➔ How can we print the text, “Amit_Data_Science_Diploma”in Python?

- print Amit_Data_Science_Diploma
- Print ” Amit_Data_Science_Diploma”
- Print(“Amit_Data_Science_Diploma”)
- Print(Amit_Data_Science_Diploma)

AMIT

Quiz(Solution):

➔ How can we print the text, “Amit_Data_Science_Diploma”in Python?

- print Amit_Data_Science_Diploma
- Print ” Amit_Data_Science_Diploma”
- Print(“Amit_Data_Science_Diploma”)
- Print(Amit_Data_Science_Diploma)

AMIT

Quiz:

➔ What will be printed by the following code?

- One
  Two
  Three
- One
  Three
- Two
- three

```
# print ("One")
print("Two")
# Three
```


AMIT

Quiz(Solution):

➔ What will be printed by the following code?

- One
  Two
  Three
- One
  Three
- Two
- three

```
# print ("One")
print("Two")
# Three
```


AMIT

Variables:

- ➔ A variable is simply a name to which a value can be assigned.
- ➔ Variables allow us to give meaningful names to data.
- ➔ The simplest way to assign a value to a variable is through the = operator.
- ➔ A big advantage of variables is that they allow us to store data so that we can use it later to perform operations in the code.
- ➔ Variables are mutable. Hence, the value of a variable can always be updated or replaced.

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
z = None               # A null value
```


AMIT

PYTHON STATEMENTS AND VARIABLES

Python statements means a logical line of code that can be either expression or assignment statement.

Expression means a sequence of numbers, strings, operators and objects that logically can be valid for executing.

Simple assignment statement means a statement with its R.H.S just a value-based expression or a variable or an operation.

Augmented assignment statement means a statement where the arithmetic operator is combined in the assignment.

Notes:

- A statement can be written in multi-lines by using \ character.
- Multiple statements can be written in same line with ; separator.

AMIT

Variables:

➔ Variables can change type, simply by assigning them a new value of a different type.

```
x = 1
x = "string value"
```

➔ Python allows you to assign a single value to several variables simultaneously.

```
a = b = c = 1
```

➔ You can also assign multiple objects to multiple variables.

```
a, b, c = 1, 2, "john"
```

➔ Variables are Case-Sensitive:
x is different from X.

PYTHON IDENTIFIERS

Python identifiers means the user-defined name that is being given to anything in your code as the variables, function, class or any other object in your code.

Guidelines for creating a python identifiers:

1- Use any sequence of lower case (a - z) or upper case (A - Z) letters in addition to numbers (0 - 9) and underscores (_).

2- No special characters or operators are allowed. (,), [,], #, $, %, !, @, ~, &, +, =, -, /, \. 3 - **Don't** start your identifier with a number as it is not valid.

4- Some keywords are reserved as False, True, def, del, if, for, raise, return, None, except, lambda, with, while, try, class, continue, as, assert, elif, else, is, in, import, not, from global , pass, finally and yield. You can't name an identifier with these words on their own however you can use them as sub-name such as True_stat or def_cat

5 - Make the user-defined name meaningful.

AMIT

PYTHON IDENTIFIERS

6 - Python is case sensitive language so variable1 identifier is different from Variable1 identifier..

Identifiers or user-defined names has a convention in python which is as follows:

- Never use l “lower-case el” or I “Upper-case eye” or O “Upper-case Oh” as single character variable name as in some fonts they are indistinguishable

Class names	Variable names / Methods / Functions / Arguments / Globals	Constants
PascalCase	snake_case	FULLY CAPITALIZED

AMIT

Variables:

➔ Do not need type definition when creating a variable.

In C Langauge for example, to declare a variable:

> int x = 4

In python, no need for declaring the type ‘int’

> x = 4

It knows automatically that x is an integer.

PYTHON DATA TYPES

Data type means the format that decides the shape and bounds of the data. In python, we don't have to explicitly predefine the variable data type but it is a dynamic typing technique. Dynamic typing means that the interpreter knows the data type of the variable at the runtime from the syntax itself.

The data types in python can be classified into:

1- Numbers
2 - Booleans
3 - Strings
4 - Bytes
5 - Lists
6 - Arrays
7- Tuples
8 - Sets
9 - Dictionaries

AMIT

PYTHON DATA TYPES

Any variable in python carries an instance of object which can be mutable or immutable. When an object is created it takes a unique object id that can be check by passing the variable to the built-in function *id()* . The type of the object is defined at runtime as mentioned before.

Immutable objects can’t be changed after it is created as int, float, bool, string, unicode, tuple.

Mutable objects can be changed after it is created as list, dict, set and user-defined classes.

AMIT

Numbers:

➔ Python is one of the most powerful languages when it comes to manipulating numerical data.

➔ There are three built-in data types for numbers in Python:

- Integer (int)
- Floating-point numbers (float)
- Complex numbers: <real part> + <imaginary part>j (not used much in Python programming)

Numbers:

- ➔ The integer data type is comprised of all the positive and negative whole numbers.
- ➔ Floating-point numbers, or floats, refer to positive and negative decimal numbers.
- ➔ Python allows us to create decimals up to a very high decimal place.
- ➔ This ensures accurate computations for precise values.
- ➔ Python also supports complex numbers, or numbers made up of a real and an imaginary part.

```
print(complex(10, 20))  # Represents the complex number (10 + 20j)
print(complex(2.5, -18.2))  # Represents the complex number (2.5 - 18.2j)
```

```
1  print(10)  # A positive integer
2  print(-3000)  # A negative integer
3
4  num = 123456789  # Assigning an integer to a variable
5  print(num)
6  num = -16000  # Assigning a new integer
7  print(num)
```


AMIT

PYTHON DATA TYPES: NUMBERS

To get the infinity value or the nan values, you can pass their words to float() as follows:

For nan values (Not a number - indefinite form):

float('nan')

float('NaN')

For infinity values:

float('inf')

float('InF')

float('InFiNiy)

float('infinity')

To check other math built-in functions in python:

https://docs.python.org/2/library/math.html

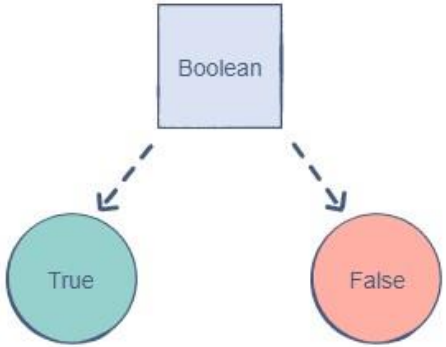
AMIT

Booleans:

- ➔ The Boolean (also known as bool) data type allows us to choose between two values: true and false.
- ➔ In Python, we can simply use True or False to represent a bool

Note: The first letter of a bool needs to be capitalized in Python.

- ➔ A Boolean is used to determine whether the logic of an expression or a comparison is correct. It plays a huge role in data comparisons.



AMIT

PYTHON DATA TYPES: BYTES

Byte is an immutable data type which can store sequence of bytes (each 8-bits) ranging from 0 to 255.

Differences between strings and bytes:

- Byte object contains sequence of bytes while String contains sequence of characters.
- Bytes can be read by machines while strings can be read by humans.
- Bytes is stored directly on machines while strings needs to be encoded first.

Mostly we stick in dealing with strings rather than using byte objects.

AMIT

Common Number Functions:

Function	Description
int(x)	to convert x to an integer
float(x)	to convert x to a floating-point number
abs(x)	The absolute value of x
cmp(x,y)	-1 if x < y, 0 if x == y, or 1 if x > y
sqrt(x)	The square root of x for x > 0
log(x)	The natural logarithm of x, for x> 0
pow(x,y)	The value of x**y

Strings:

- ➔ A string is a collection of characters closed within single or double quotation marks (immutable).
- ➔ You can update an existing string by (re)assigning a variable to another string.
- ➔ Python does not support a character type; these are treated as strings of length one.

```
>>> str= "strings are immutable!"
>>> str[0]="S"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

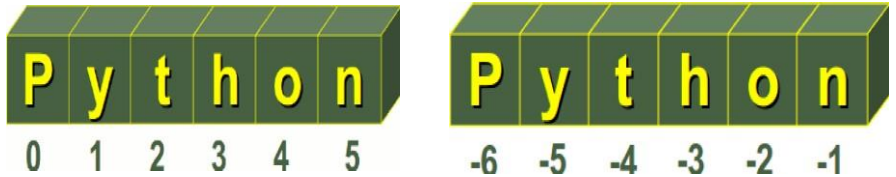

AMIT

Strings:

➔ Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals.

➔ String indexes starting at 0 in the beginning of the string and working their way from -1 at the end

```
name1 = "sample string"
name2 = 'another sample string'
name3 = """a multiline
  string example"""
```



AMIT

Strings Indexing:

➔ You can index a string (access a character) by using square brackets:

```
1  batman = "Bruce Wayne"
2
3  first = batman[0]  # Accessing th
4  print(first)
5
6  space = batman[5]  # Accessing th
7  print(space)
8
```

```
1  batman = "Bruce Wayne"
2  print(batman[-1])  # Corresponds to batman[10]
3  print(batman[-5])  # Corresponds to batman[6]
```


AMIT

Strings Slicing:

➔ Slicing is the process of obtaining a portion (substring) of a string by using its indices. Given a string, we can use the following template to slice it and obtain a substring String [start:end]

- start is the index from where we want the substring to start.
- end is the index where we want our substring to end.

➔ The character at the end index in the string, will not be included in the substring obtained through this method.

```
1  my_string = "This is MY string!"
2  print(my_string[0:4]) # From the start till before the 4th index
3  print(my_string[1:7])
4  print(my_string[8:len(my_string)]) # From the 8th index till the end
```

Output

```
This
his is
MY string!
```


AMIT

Strings Slicing with a step:

- ➔ Until now, we've used slicing to obtain a contiguous piece of a string, i.e., all the characters from the starting index to before the ending index are retrieved.
- ➔ However, we can define a step through which we can skip characters in the string. The default step is 1, so we iterate through the string one character at a time.
- ➔ The step is defined after the end index

string[start:end:step]

```
1  my_string = "This is MY string!"
2  print(my_string[0:7])  # A step of 1
3  print(my_string[0:7:2])  # A step of 2
4  print(my_string[0:7:5])  # A step of 5
5
```

Output

```
This is
Ti s
Ti
```


AMIT

Strings Reverse Slicing:

➔ Strings can also be sliced to return a reversed substring. In this case, we would need to switch the order of the start and end indices.

➔ A negative step must also be provided The step is defined after the end index

```
1  my_string = "This is MY string!"
2  print(my_string[13:2:-1]) # Take 1 step back each time
3  print(my_string[17:0:-2]) # Take 2 steps back. The opposite of what happens in the slide above
4
```

Output

```
rts YM si s
!nrsY ish
```


AMIT

Strings Partial Slicing:

- ➔ One thing to note is that specifying the start and end indices is optional.
- ➔ If start is not provided, the substring will have all the characters until the end index.
- ➔ If end is not provided, the substring will begin from the start index and go all the way to the end.

```
1  my_string = "This is MY string!"
2  print(my_string[:8])  # All the characters before 'M'
3  print(my_string[8:])  # All the characters starting from 'M'
4  print(my_string[:])  # The whole string
5  print(my_string[::-1])  # The whole string in reverse (step is -1)
6
```

Output

```
This is
MY string!
This is MY string!
!gnirts YM si sihT
```


AMIT

Quiz:

➔ What is the output of the following code?

```
> my_string = "0123456789"
> print(my_string[-2: -6: -2])
```

- 5432
- 8765
- 532
- 86

Quiz(Solution):

➔ What is the output of the following code?

```
> my_string = "0123456789"
> print(my_string[-2: -6: -2])
```

- 5432
- 8765
- 532
- 86

AMIT

Quiz:

➔ String indices can be floats?

- True
- False

Quiz(Solution):

➔ String indices can be floats?

- True
- False

AMIT

Common String Operators:

➔ Assume string variable a holds 'Hello' and variable b holds 'Python'

Operator	Description	Example
+	**Concatenation** - Adds values on either side of the operator	a + b will give **HelloPython**
*	**Repetition** - Creates new strings, concatenating multiple copies of the same string	a*2 will give **HelloHello**
[ ]	**Slice** - Gives the character from the given index	a[1] will give **e** a[-1] will give **o**
[:]	**Range Slice** - Gives the characters from the given range	a[1:4] will give **ell**
in	**Membership** - Returns true if a character exists in the given string	'H' in a will give **True**

Common String Operators:

➔ Strings are an example of Python objects. An object contains both data (the actual string itself) and methods, which are effectively functions that are built into the object and are available to any instance of the object.

➔ upper() Return a capitalized version of string
➔ lower() Return a copy of the string converted to lowercase.
➔ find() searches for the position of one string within another
➔ strip() removes white space (spaces, tabs, or newlines) from the beginning and end of a string

Other useful methods in this link

AMIT

Python User Input from Keyboard

- ➔ Python user input from the keyboard can be read using the input() built-in function.
- ➔ The input from the user is read as a string and can be assigned to a variable.
- ➔ After entering the value from the keyboard, we have to press the "Enter" button. Then the input() function reads the value entered by the user.

```
In [*]: input("please enter a value")
```

please enter a value

AMIT

Python User Input from Keyboard

- ➔ The program halts indefinitely for the user input. There is no option to provide timeout value.
- ➔ The syntax of input() function is: input(prompt)
- ➔ The prompt string is printed on the console and the control is given to the user to enter the value. You should print some useful information to guide the user to enter the expected value.

Python User Input from Keyboard

➔ What is the type of user entered value?
The user entered value is always converted to a string and then assigned to the variable. Let's confirm this by using type() function to get the type of the input variable.

```
In [12]: my_input=input("please enter a number")
         print(type(my_input))

         please enter a number10
         <class 'str'>
```

➔ How to get an Integer as the User Input?
There is no way to get an integer or any other type as the user input. However, we can use the built-in functions to convert the entered string to the integer.

```
In [13]: my_input=input("please enter a number")
         my_input=int(my_input)
         print(type(my_input))

         please enter a number100
         <class 'int'>
```


AMIT

Any Questions?

THANK YOU!