DATA ANALYTICS WITH PYTHON PROGRAMMING

MKU Reasearch Team
Dr.S.JOTHILAKSHMI, M.Sc, M.Phil, PhD.
The American College,
Madurai

Python

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- It was created by Guido van Rossum during 1985- 1990.
- Monty Python's Flying Circus

WHAT IS PYTHON?

- Python is a general purpose, dynamic, high-level, and interpreted programming language.
- It supports Object Oriented programming approach to develop applications.
- Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.

- Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.

- Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.

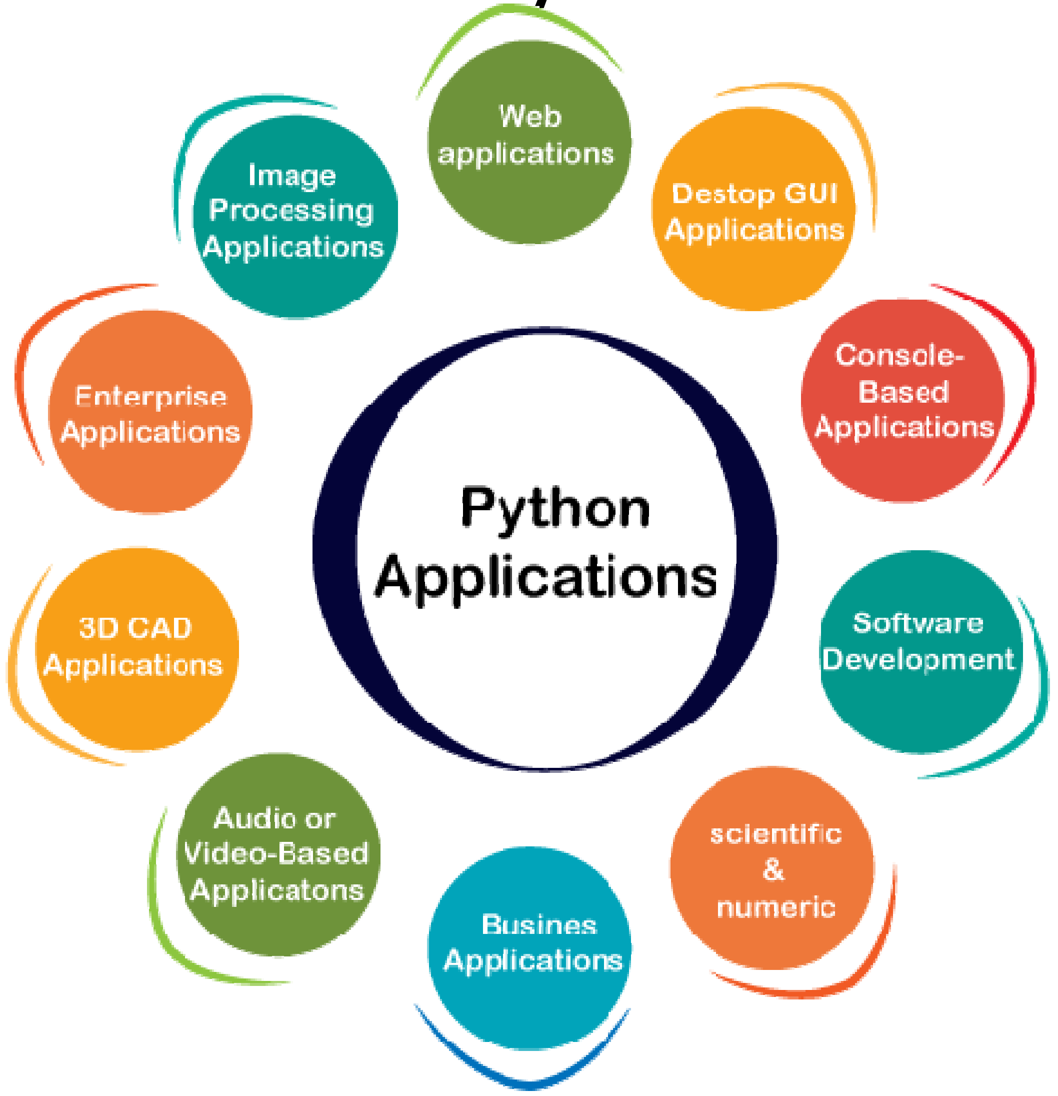
- it is *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.

- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java

WHY LEARN PYTHON?

- Easy to use and Learn
- Expressive Language
- Interpreted Language
- Object-Oriented Language
- Open Source Language
- Extensible
- Learn Standard Library
- GUI Programming Support
- Integrated
- Embeddable
- Dynamic Memory Allocation
- Wide Range of Libraries and Frameworks

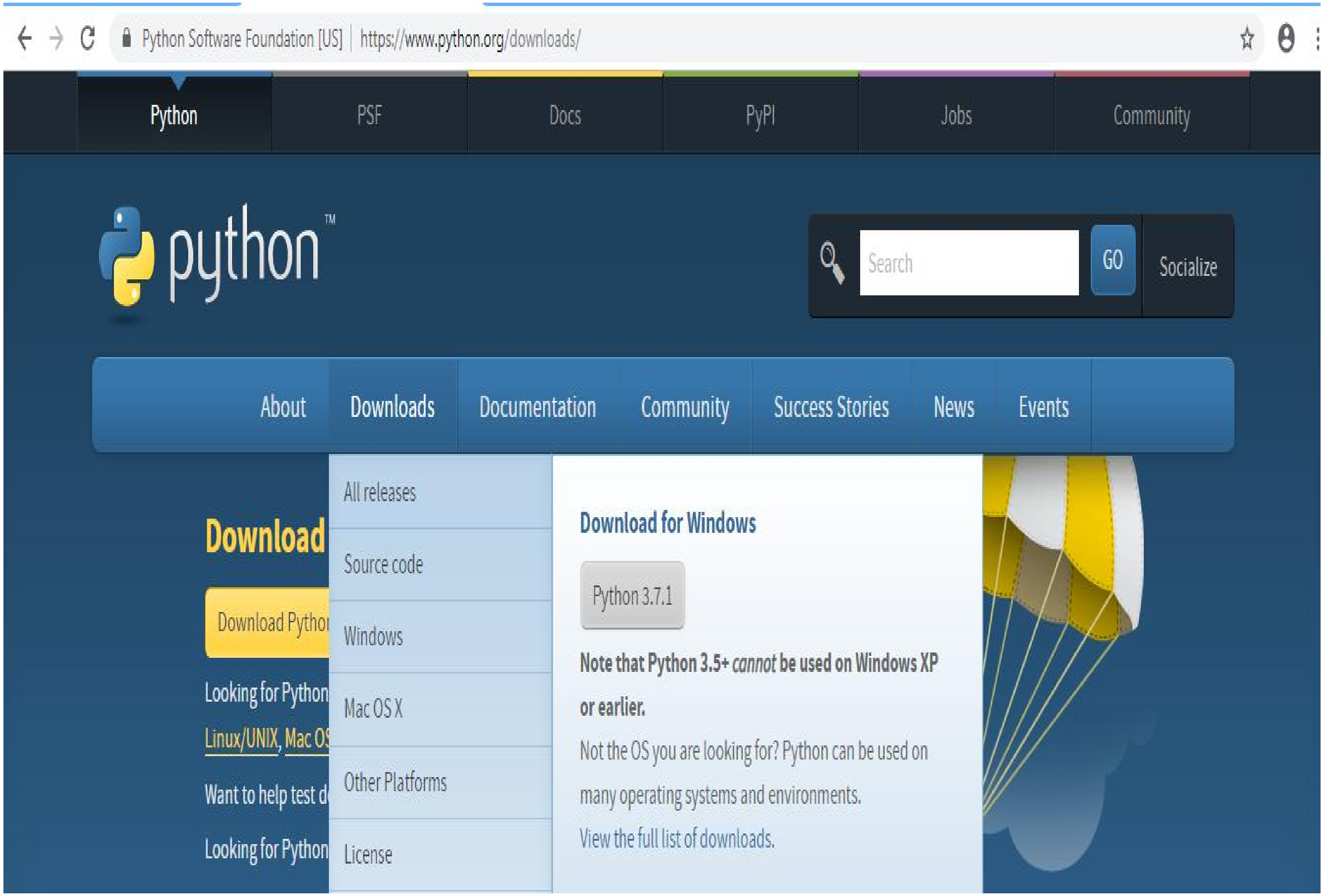
Where is Python used?



Installation

Windows Installation

- Installation on Windows
- Visit the link *https://www.python.org/downloads/* to download the latest release of Python. In this process, we will install Python 3.8.6 on our Windows operating system. When we click on the above link, it will bring us the following page.



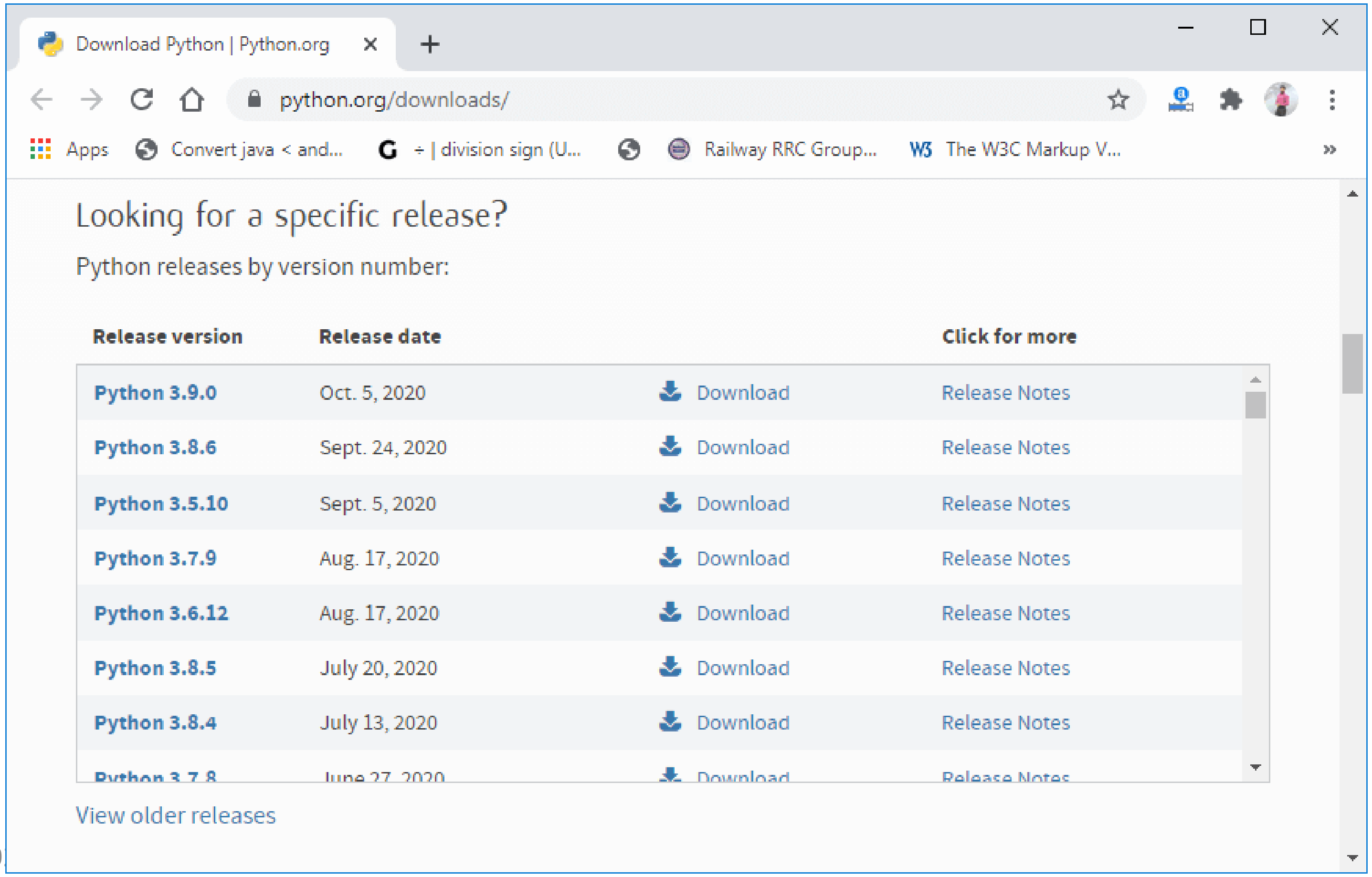
Download for Windows

Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

- **Step - 1: Select the Python's version to download.**
- Click on the download button.



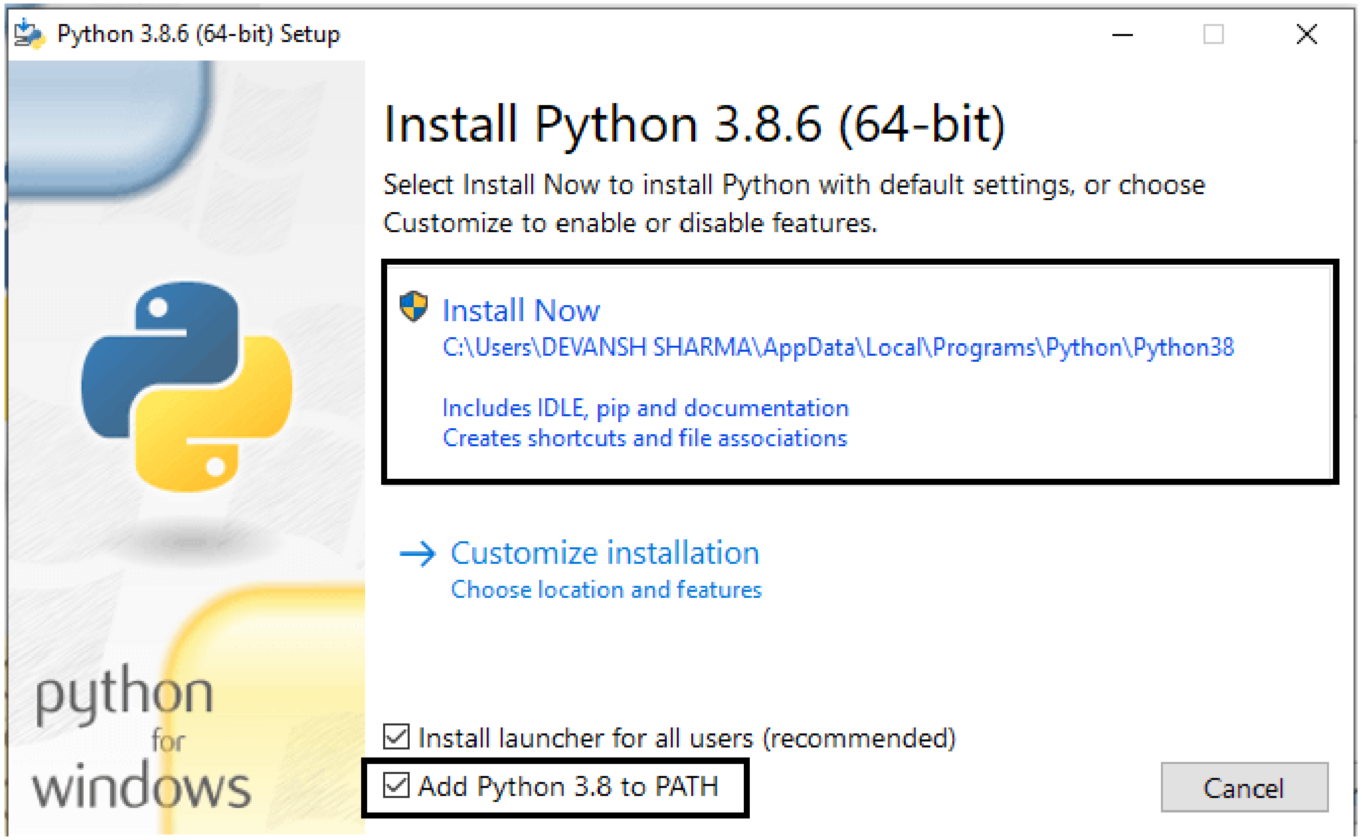
Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.9.0	Oct. 5, 2020	Download	Release Notes
Python 3.8.6	Sept. 24, 2020	Download	Release Notes
Python 3.5.10	Sept. 5, 2020	Download	Release Notes
Python 3.7.9	Aug. 17, 2020	Download	Release Notes
Python 3.6.12	Aug. 17, 2020	Download	Release Notes
Python 3.8.5	July 20, 2020	Download	Release Notes
Python 3.8.4	July 13, 2020	Download	Release Notes
Python 3.7.8	June 27, 2020	Download	Release Notes

View older releases

- **Step - 2: Click on the Install Now**
- Double-click the executable file, which is downloaded; the following window will open. Select Customize installation and proceed. Click on the Add Path check box, it will set the Python path automatically.



Python 3.8.6 (64-bit) Setup

Install Python 3.8.6 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

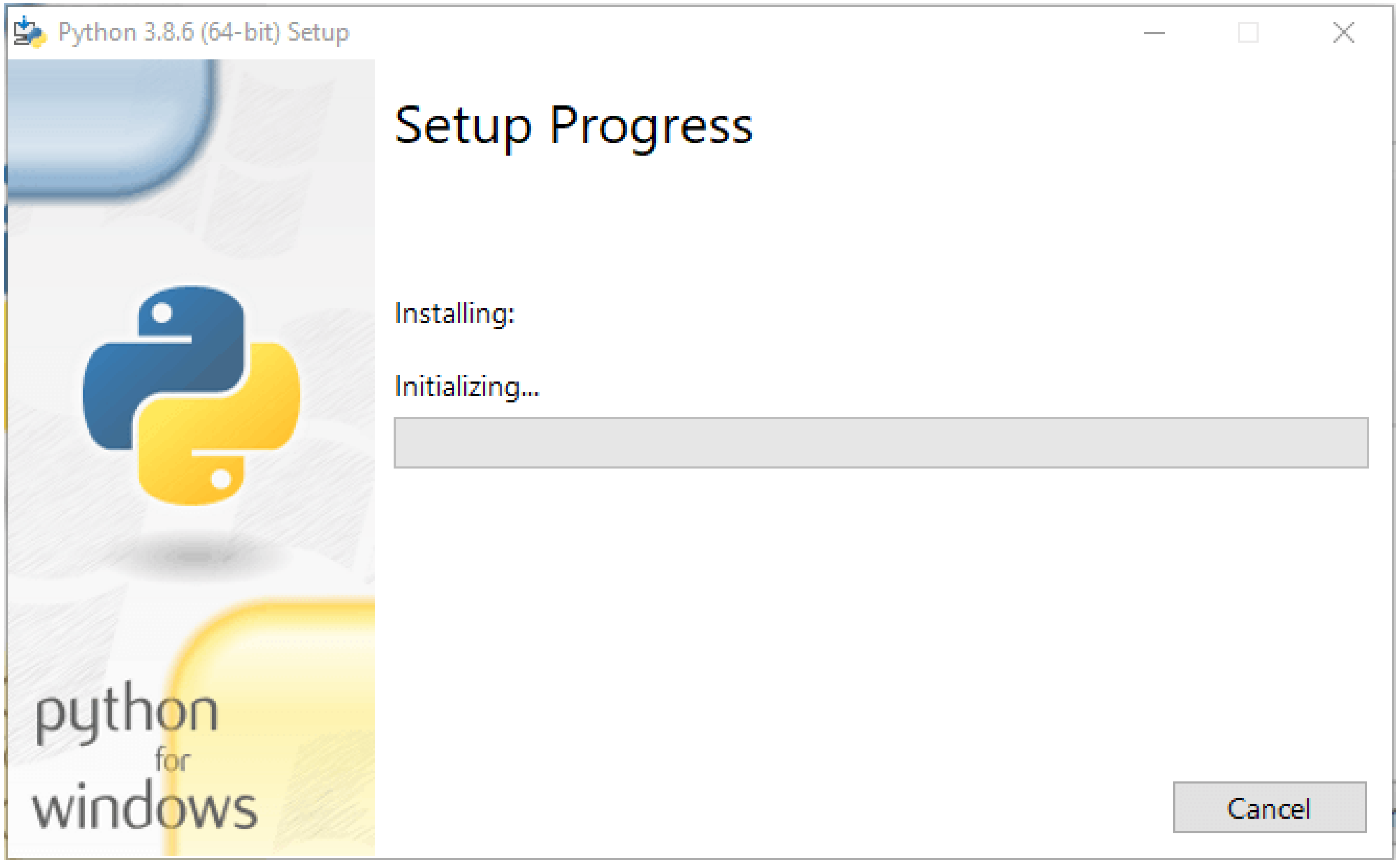
Install Now
C:\Users\DEVANSH SHARMA\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.8 to PATH

- **Step - 3 Installation in Process**

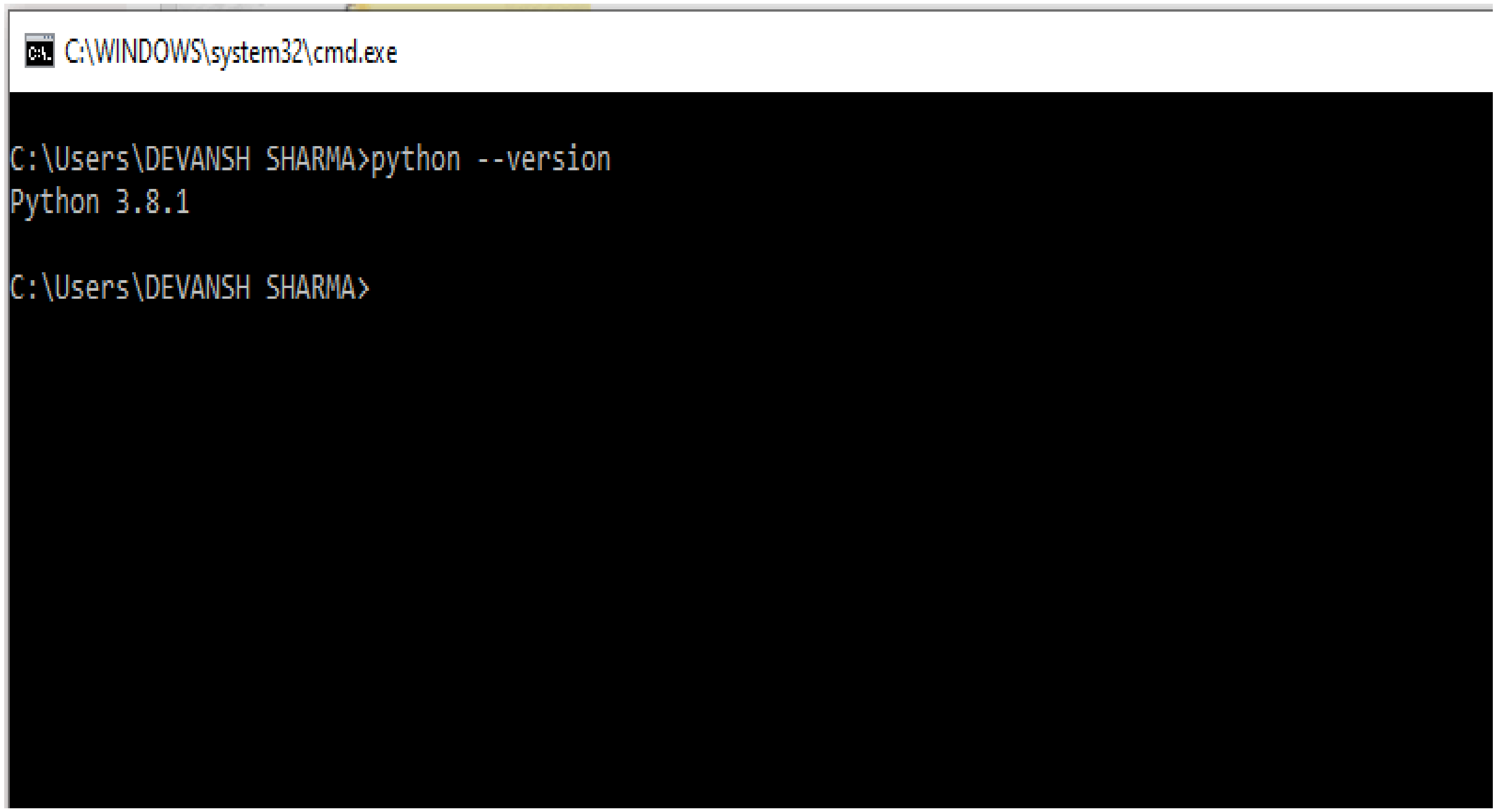

Python 3.8.6 (64-bit) Setup

Setup Progress

Installing:

Initializing...

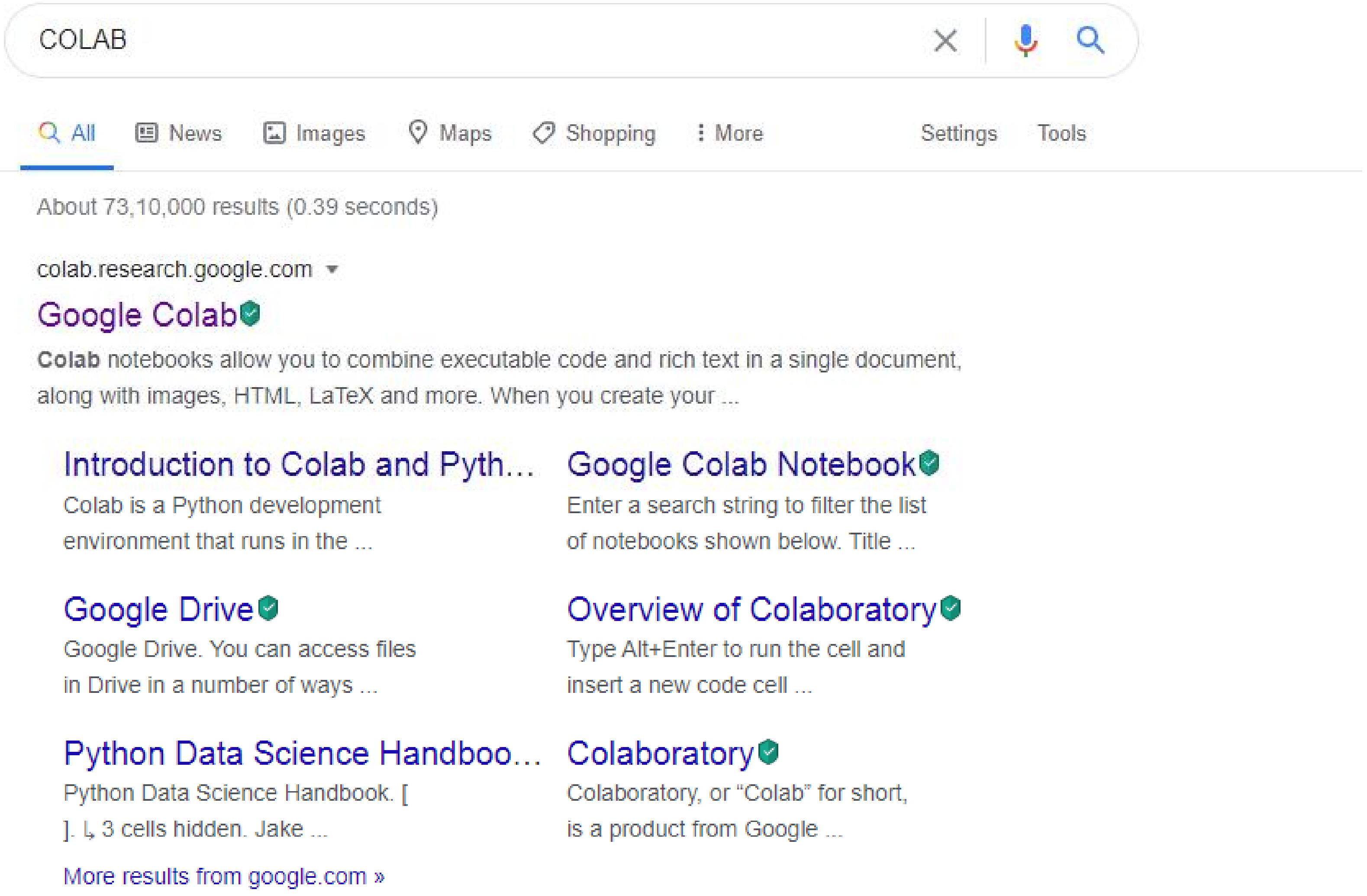
Now, try to run python on the command prompt. Type the command python -version in case of python3.


C:\WINDOWS\system32\cmd.exe

```
C:\Users\DEVANSH SHARMA>python --version
Python 3.8.1

C:\Users\DEVANSH SHARMA>
```

We are ready to work with the Python.



About 73,10,000 results (0.39 seconds)

colab.research.google.com

Google Colab

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your ...

Introduction to Colab and Pyth...
Colab is a Python development environment that runs in the ...

Google Colab Notebook
Enter a search string to filter the list of notebooks shown below. Title ...

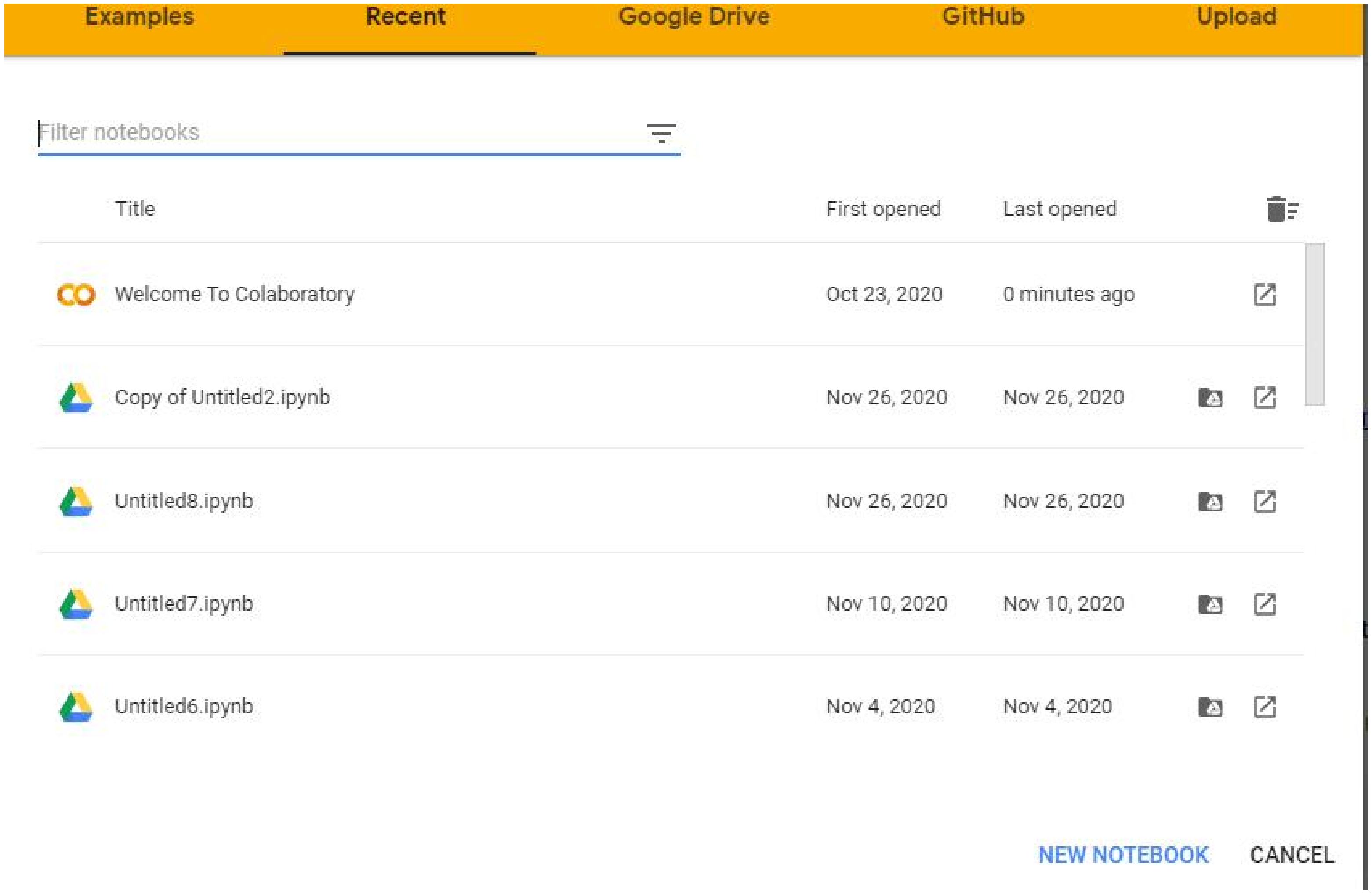
Google Drive
Google Drive. You can access files in Drive in a number of ways ...

Overview of Colaboratory
Type Alt+Enter to run the cell and insert a new code cell ...

Python Data Science Handboo...
Python Data Science Handbook. [ ]. ↳ 3 cells hidden. Jake ...

Colaboratory
Colaboratory, or "Colab" for short, is a product from Google ...

More results from google.com »



Title	First opened	Last opened
Welcome To Colaboratory	Oct 23, 2020	0 minutes ago
Copy of Untitled2.ipynb	Nov 26, 2020	Nov 26, 2020
Untitled8.ipynb	Nov 26, 2020	Nov 26, 2020
Untitled7.ipynb	Nov 10, 2020	Nov 10, 2020
Untitled6.ipynb	Nov 4, 2020	Nov 4, 2020

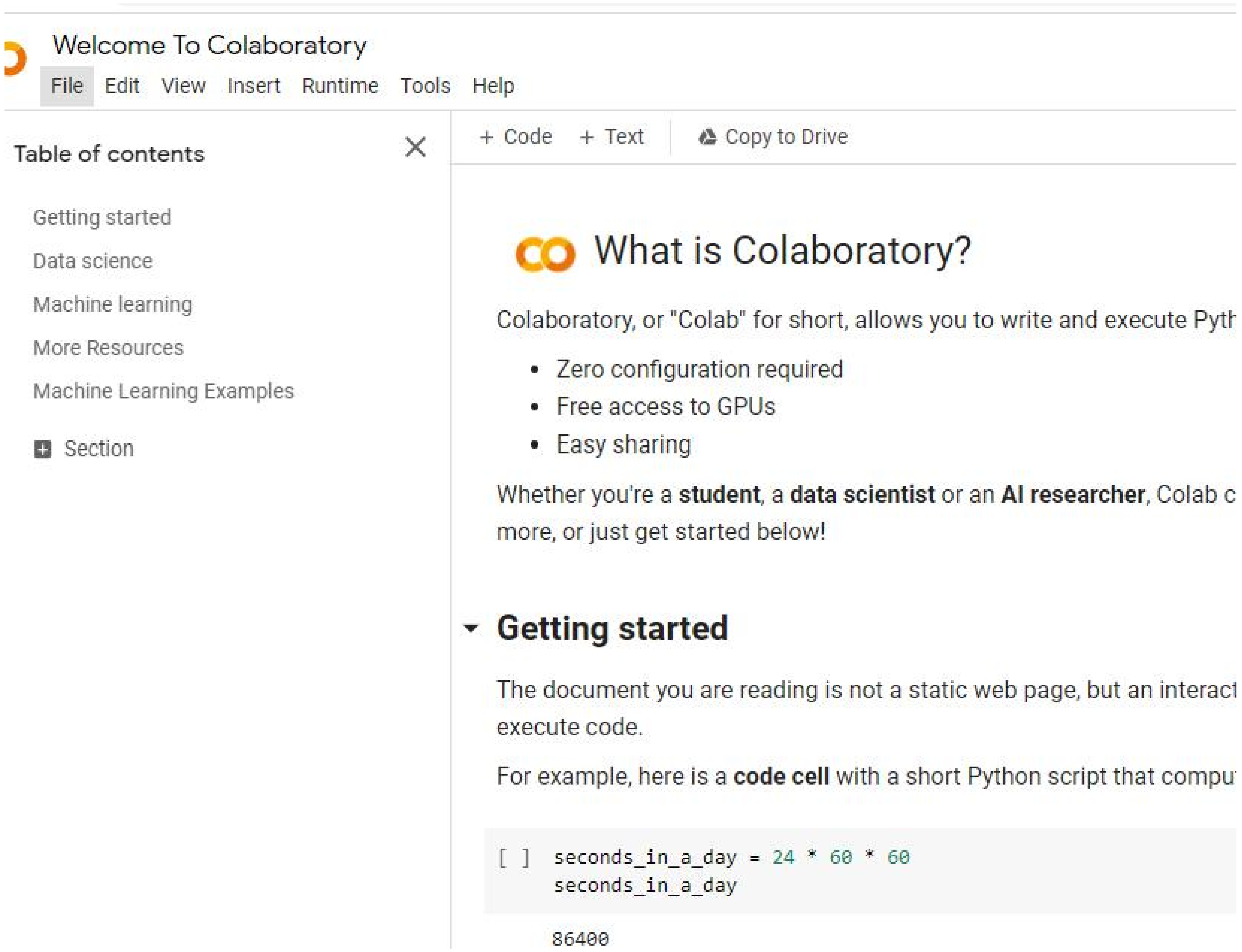

Welcome To Colaboratory
File Edit View Insert Runtime Tools Help
+ Code + Text Copy to Drive

Table of contents

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Pyth

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab c more, or just get started below!

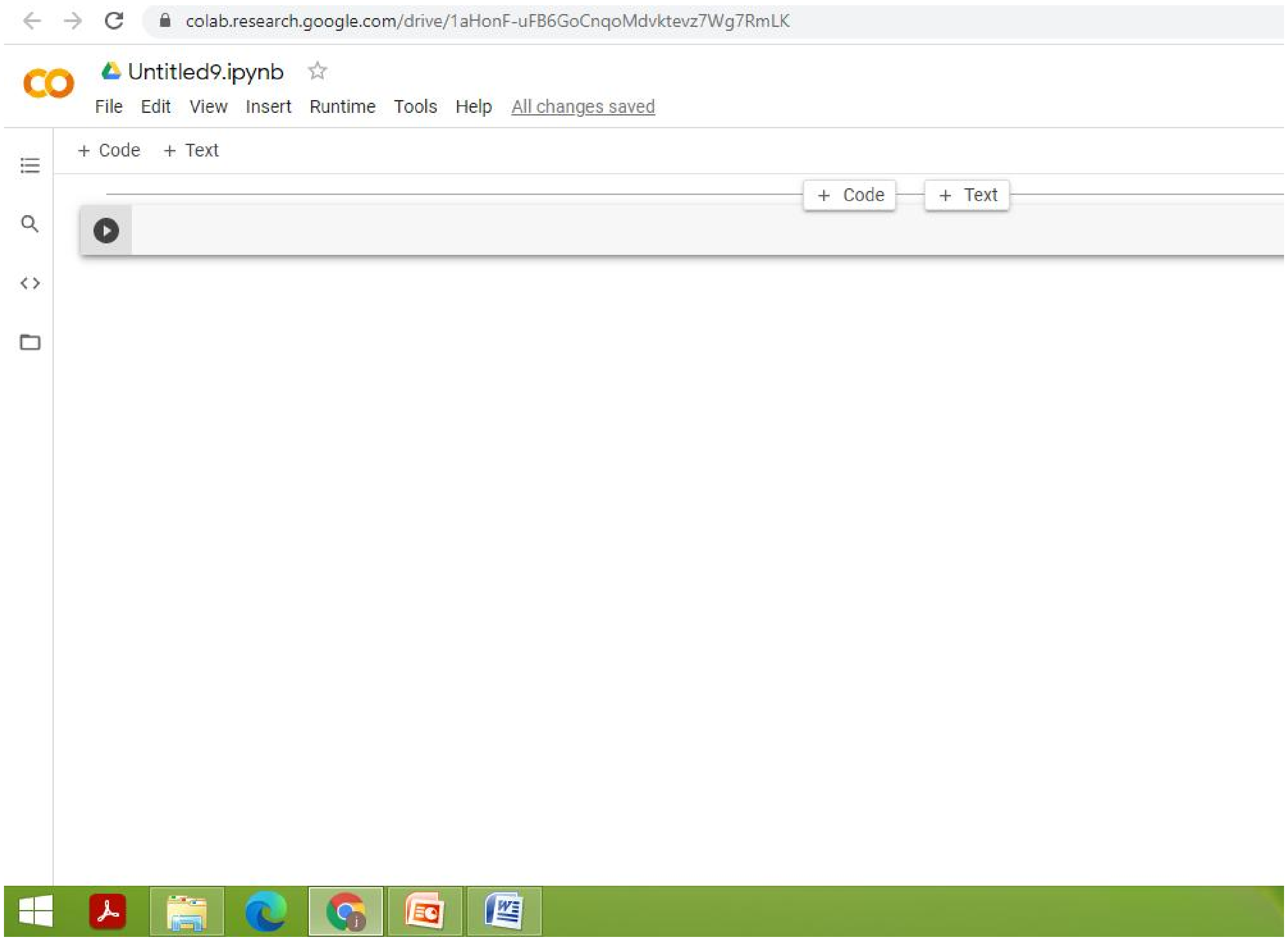
Getting started

The document you are reading is not a static web page, but an interact execute code.

For example, here is a **code cell** with a short Python script that compu

```
[ ]  seconds_in_a_day = 24 * 60 * 60
     seconds_in_a_day

     86400
```


colab.research.google.com/drive/1aHonF-uFB6GoCnqoMdvktevz7Wg7RmLK
Untitled9.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
+ Code
+ Text
+ Code
+ Text

Two Basic modes

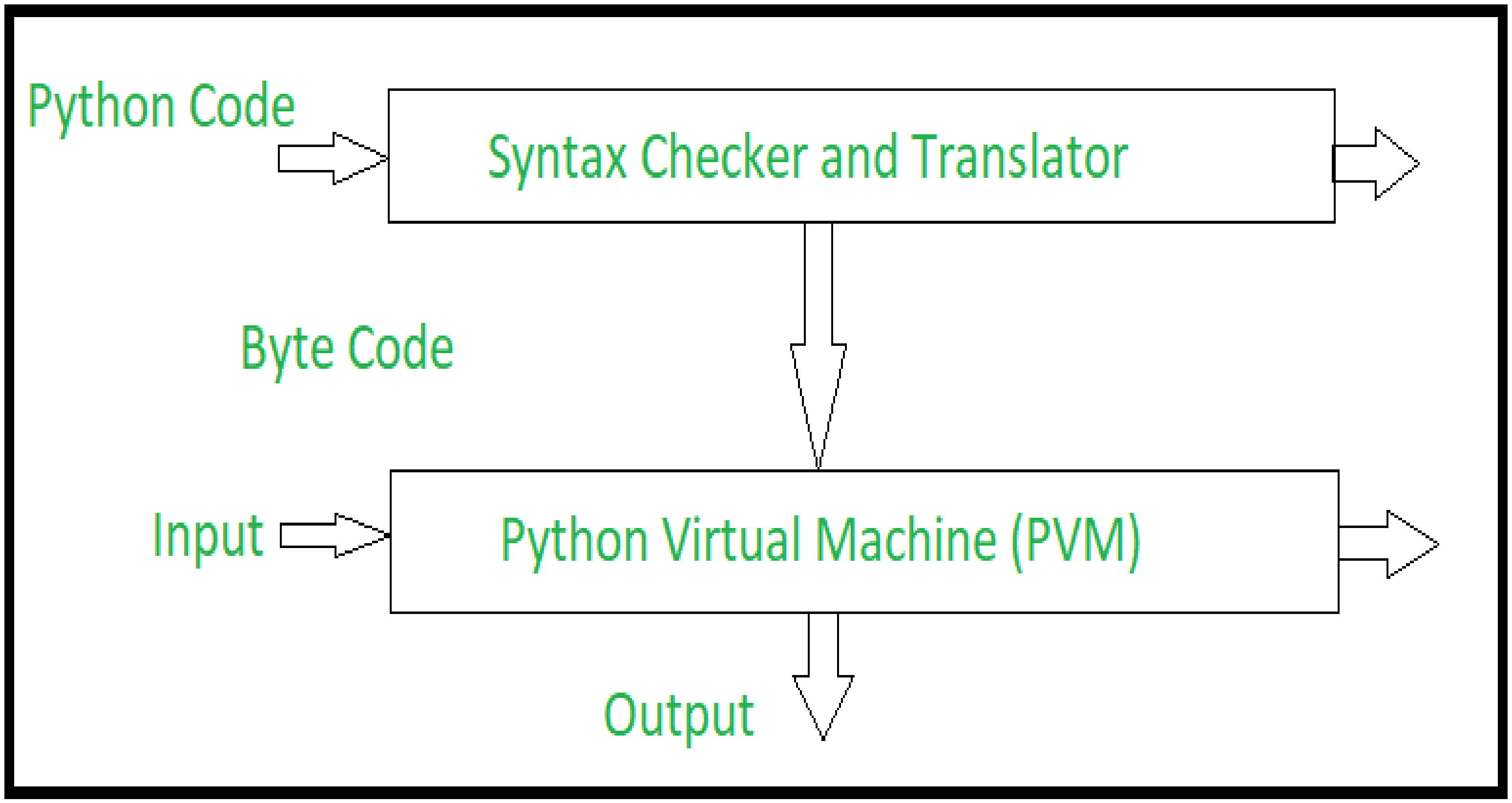
- Python has two basic modes:
  - → Script
  - → Interactive.
- The normal mode is the mode where the scripted and finished .py files are run in the Python interpreter.
- Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory.
- As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.

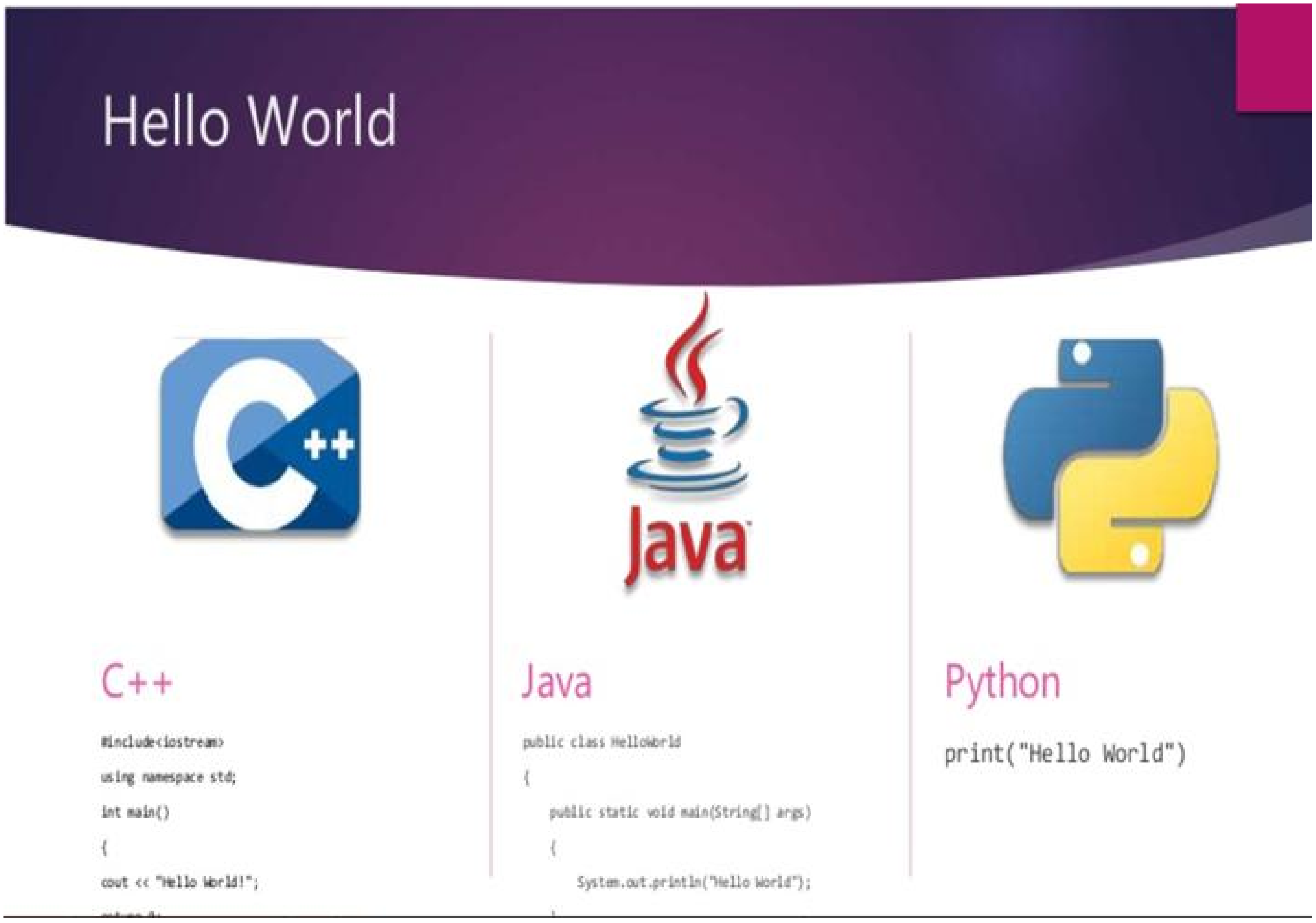
Script Mode Programming

- Let us write a simple Python program in a script.
- Python files have extension **.py**.
- Type the following source code in a test.py file:
- print "Hello, Python!"

Internal working of Python

- **The Python source code goes through the following to generate an executable code :**
- **Step 1:** The python compiler reads a python source code or instruction. Then it verifies that the instruction is well-formatted, i.e. it checks the syntax of each line. If it encounters an error, it immediately halts the translation and shows an error message.
- **Step 2:** If there is no error, i.e. if the python instruction or source code is well-formatted then the compiler translates it into its equivalent form in an intermediate language called “Byte code”.
- **Step 3:** Byte code is then sent to the Python Virtual Machine(PVM) which is the python interpreter. PVM converts the python byte code into machine-executable code. If an error occurs during this interpretation then the conversion is halted with an error message.


Python Code
Syntax Checker and Translator
Byte Code
Input
Python Virtual Machine (PVM)
Output



Hello World

C++

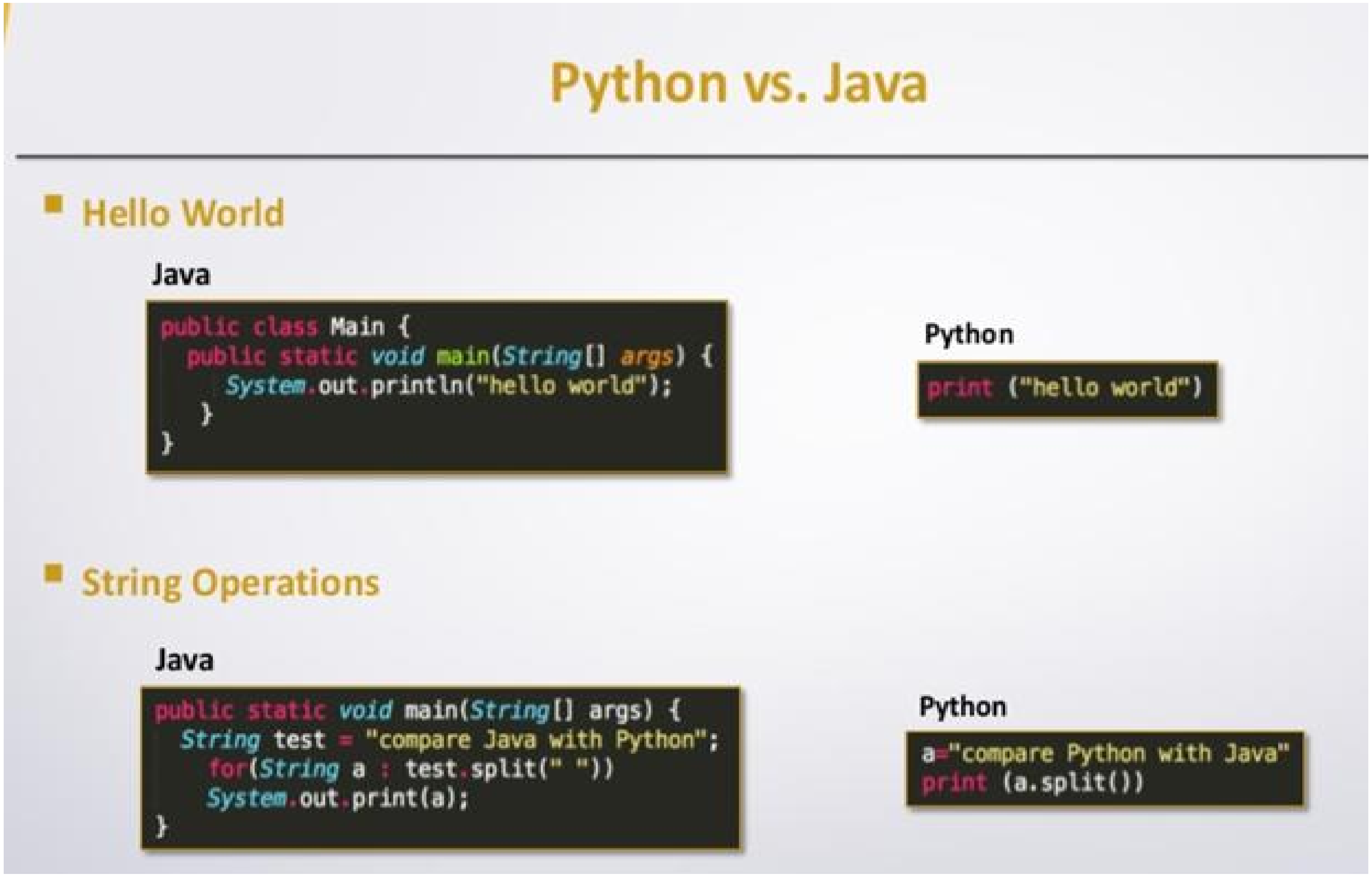
```cpp
#include<iostream>
using namespace std;
int main()
{
cout << "Hello World!";
```

Java

```java
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
```

Python

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



Python vs. Java

- **Hello World**

Java

```java
public class Main {
  public static void main(String[] args) {
     System.out.println("hello world");
   }
}
```

Python

```python
print ("hello world")
```

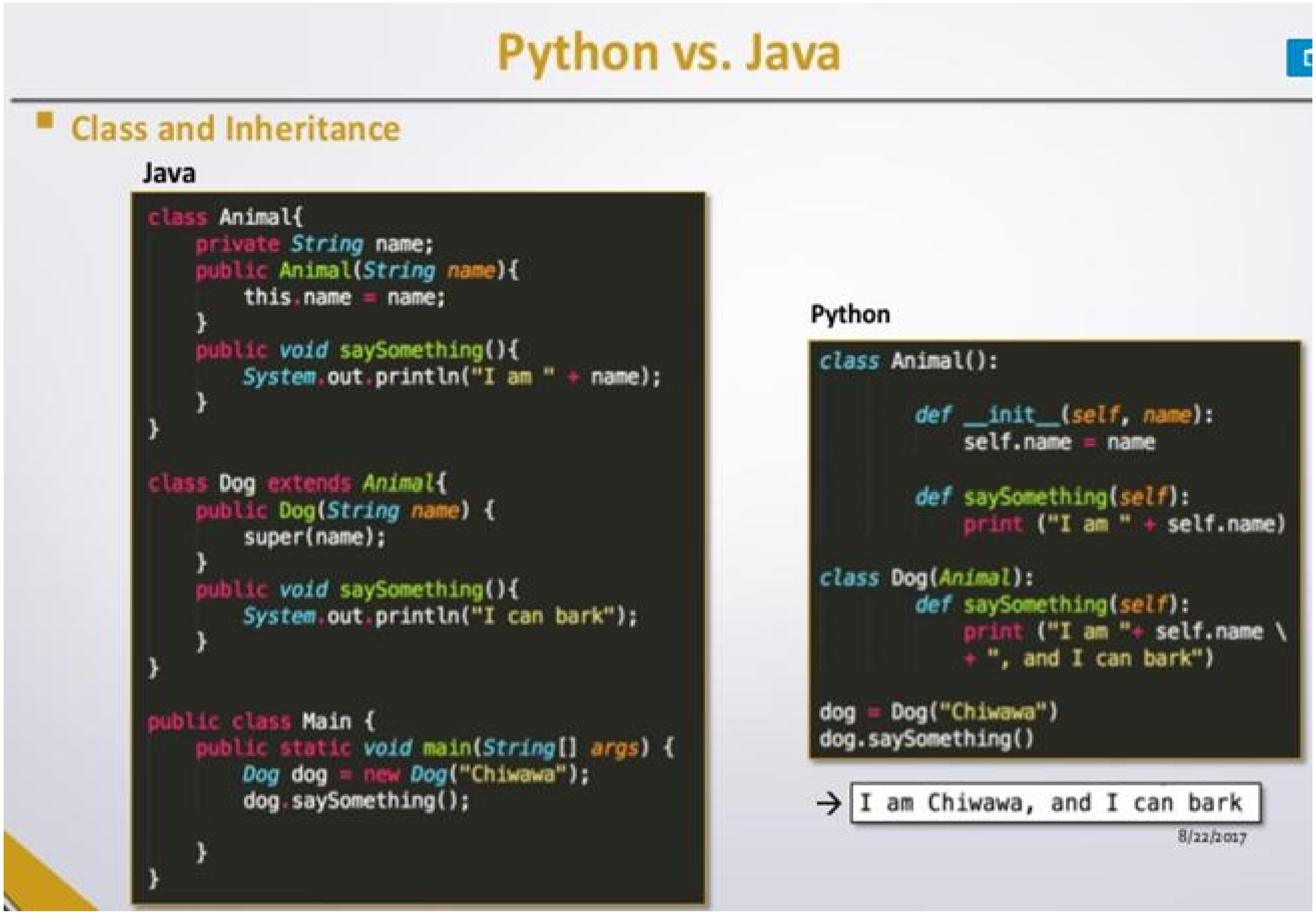
- **String Operations**

Java

```java
public static void main(String[] args) {
  String test = "compare Java with Python";
    for(String a : test.split(" "))
    System.out.print(a);
}
```

Python

```python
a="compare Python with Java"
print (a.split())
```



Python vs. Java

- Class and Inheritance

Java

```
class Animal{
    private String name;
    public Animal(String name){
        this.name = name;
    }
    public void saySomething(){
        System.out.println("I am " + name);
    }
}

class Dog extends Animal{
    public Dog(String name) {
        super(name);
    }
    public void saySomething(){
        System.out.println("I can bark");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog("Chiwawa");
        dog.saySomething();

    }
}
```

Python

```
class Animal():

        def __init__(self, name):
            self.name = name

        def saySomething(self):
            print ("I am " + self.name)

class Dog(Animal):
        def saySomething(self):
            print ("I am "+ self.name \
            + ", and I can bark")

dog = Dog("Chiwawa")
dog.saySomething()
```

→
```
I am Chiwawa, and I can bark
```

8/22/2017

Python - Variable

- Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables

Rules

- Variables are the example of identifiers. The rules to name an identifier are given below.

- **The first character of the variable must be an alphabet or underscore (_).**
- **All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).**
- **Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).**
- **Identifier name must not be similar to any keyword defined in the language.**
- **Identifier names are case sensitive; for example, my name, and MyName is not the same.**
- **Examples of valid identifiers: a123, _n, n_9, etc.**
- **Examples of invalid identifiers: 1a, n%4, n 9, etc.**

Declaring Variable and Assigning Values

- It allows us to create a variable at the required time.
- We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.
- The equal (=) operator is used to assign value to a variable.

Variable Names

- Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_)

- name = "Devansh"
- age = 20
- marks = 80.50

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously.
- For example −
- a = b = c = 1
- Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.
- You can also assign multiple objects to multiple variables.
- For example −
- a,b,c = 1,2,"john"

- Variables do not need to be declared with any particular *type*, and can even change type after they have been set.
- Example

-
  ```
  x = 4                    # x is of type int
  x = "Sally"              # x is now of type str
  print(x)
  ```

- String variables can be declared either by using single or double quotes:
- Example

-
  ```
  x = "John"
  # is the same as
  x = 'John'
  ```

- #Legal variable names:
- myvar = "John"
- my_var = "John"
- _my_var = "John"
- myVar = "John"
- MYVAR = "John"
- myvar2 = "John"

- #Illegal variable names:
- 2myvar = "John"
- my-var = "John"
- my var = "John"

Local Variable

- Local variables are the variables that declared inside the function and have scope within the function

Declaring a function

```
def add():
    # Defining local variables. They has scope only within a
function
        a = 20
        b = 30
         c = a + b
        print("The sum is:", c)

# Calling a function
        add()
```

Output:

```
The sum is: 50
```

Global Variable

- Global variables can be used throughout the program, and its scope is in the entire program If we don't use the **global** keyword, the function treats it as a local variable

Declare a variable and initialize it

```
x = 101
```

- # Global variable in function

```
def mainFunction():
   # printing a global variable
   global x
   print(x)
   # modifying a global variable
   x = 'Welcome To Javatpoint'
   print(x)
```

- mainFunction()
- **print**(x)

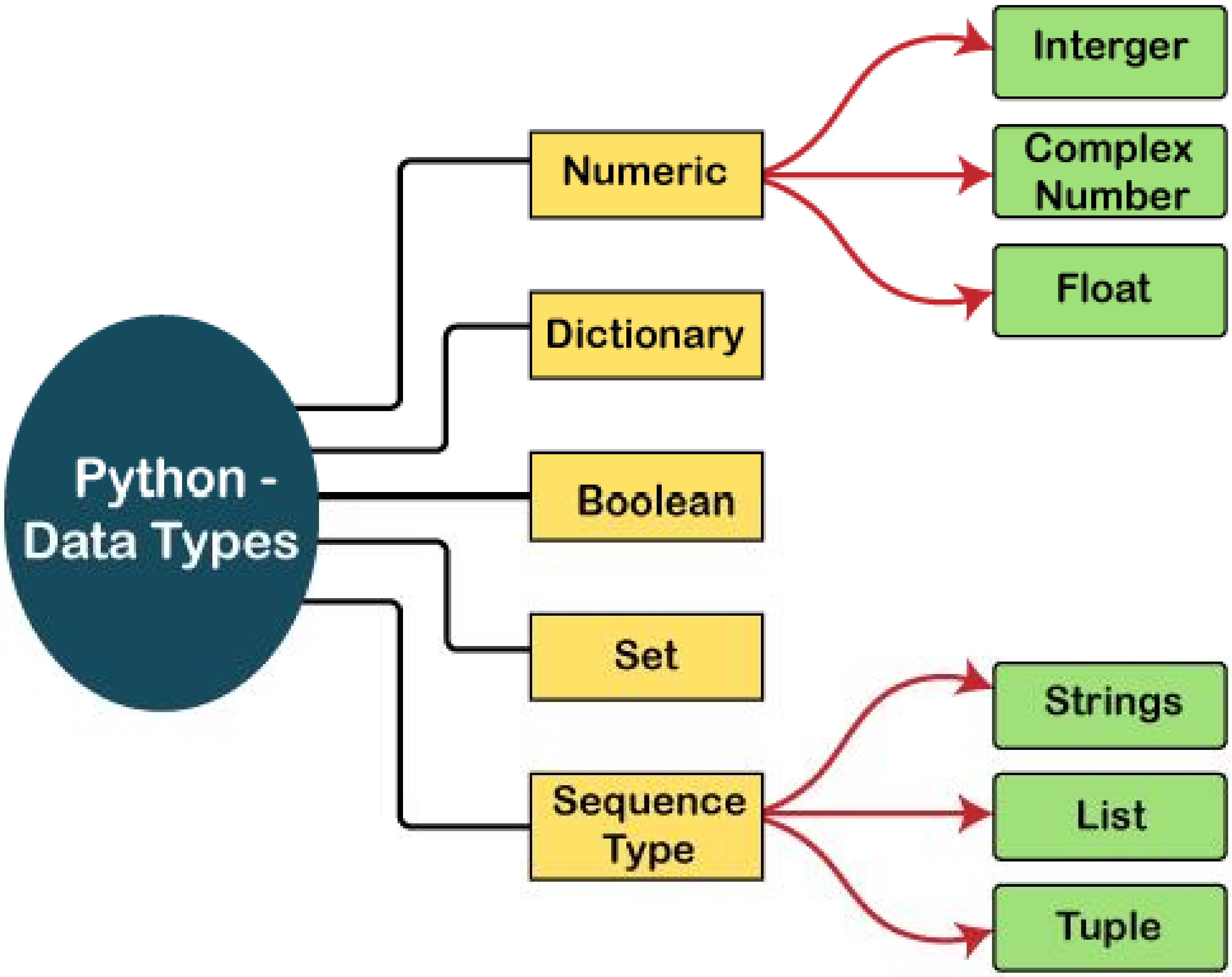
Output:

```
101
Welcome To Javatpoint
Welcome To Javatpoint
```

Delete a variable

- We can delete the variable using the **del** keyword. The syntax is given below.

- **Syntax -**
- **del** <variable_name>
- Assigning a value to x
- x = 6
- **print**(x)
- # deleting a variable.
- **del** x
- **print**(x)


Python - Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Interger
Complex Number
Float
Strings
List
Tuple

Numbers

Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type

Dictionary

Dictionary is an unordered set of a key-value pair of items

Boolean

Boolean type provides two built-in values, True and False

Set

Python Set is the unordered collection of the data type. It is iterable, mutable(can modify after creation)

Sequence Type

String

The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string

List

Python Lists are similar to arrays in C. However, the list can contain data of different types

Tuple

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data type.s A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

- Integers:
- x = int(1)   # x will be 1
- y = int(2.8) # y will be 2
- z = int("3") # z will be 3
- Floats:
- x = float(1)   # x will be 1.0
- y = float(2.8)   # y will be 2.8
- z = float("3")   # z will be 3.0
- w = float("4.2") # w will be 4.2
- Strings:
- x = str("s1") # x will be 's1'
  y = str(2)   # y will be '2'
  z = str(3.0) # z will be '3.0’

Getting the Data Type

- You can get the data type of any object by using the type() function:

- Example
- Print the data type of the variable x:

- x = 5
- print(type(x))

- Python Keywords

True	**False**	**None**	**and**	**as**
asset	**def**	**class**	**continue**	**break**
else	**finally**	**elif**	**del**	**except**
global	**for**	**if**	**from**	**import**
raise	**try**	**or**	**return**	**pass**
nonlocal	**in**	**not**	**is**	**lambda**

Lines and Indentation

- Python provides no braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.

.

- For example –

```
if True:
    print "True"
else:
    print "False"
```

Python Operators

- Operators are used to perform operations on variables and values.

- Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Python If statements

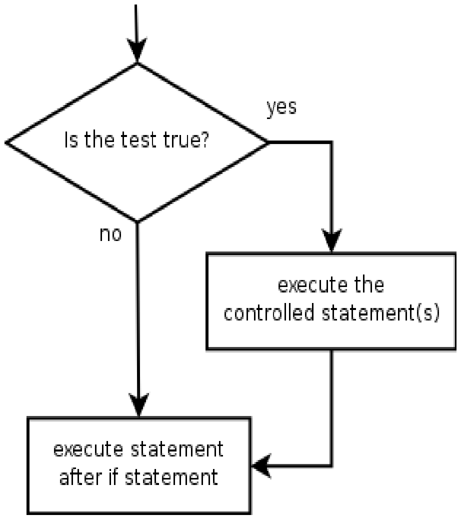
- **`if` statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
  - Syntax:

    `if` ***condition*** `:`

      statements

- Example:

```
gpa = 3.4
if gpa > 2.0:
    print "Your application is accepted."
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

if/else

- **`if/else` statement**: Executes one block of statements if a certain condition is True, and a second block of statemer
is False.

  – Syntax:

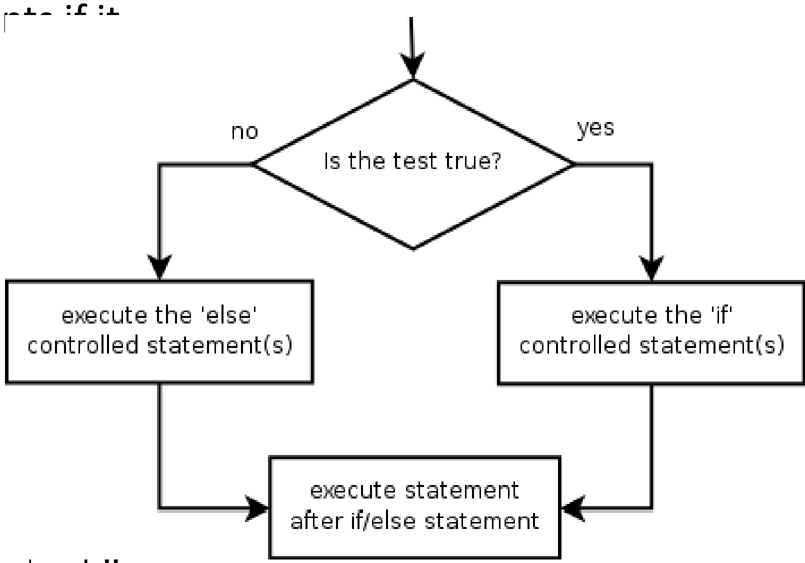
```
if condition:
    statements
else:
    statements
```

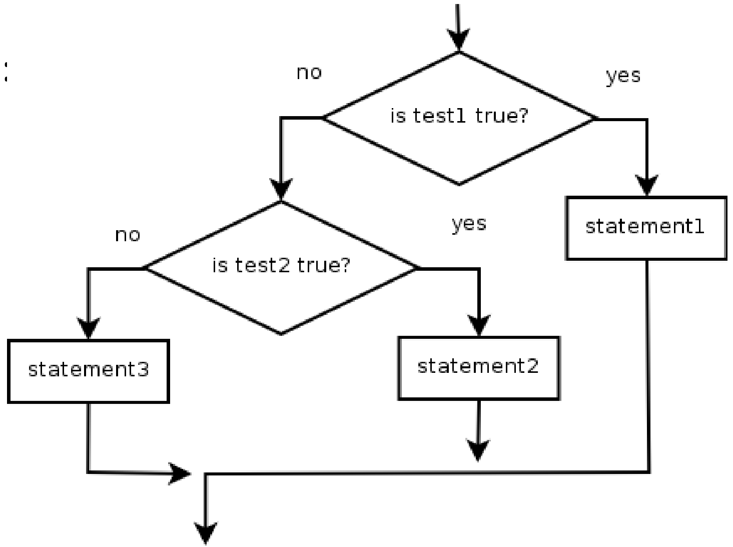
- Example:

```
gpa = 1.4
if gpa > 2.0:
    print "Welcome to Mars University!"
else:
    print "Your application is denied."
```

- Multiple conditions can be chained with `elif` ("else if"):

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```


no
yes
Is the test true?
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement



The for loop

- The syntax of for loop in python is given below.
- **for** iterating_var **in** sequence:
-     statement(s)

Example-1: Iterating string using for loop

```
str = "Python"
for i in str:
print(i)
```

Output:

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

`range`

- The `range` function specifies a range of integers:
  - `range(`***start*****,** ***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)
  - It can also accept a third value specifying the change between values.
    - `range(`***start*****,** ***stop*****,** ***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

Example:

```
for x in range(5, 0, -1):
    print x
print "Blastoff!"
```

Output:

```
5
4
3
2
1
Blastoff!
```

Nested for loop

- Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

```
 for iterating_var1 in sequence:  #outer loop
 for iterating_var2 in sequence:  #inner loop
 #block of statements
#Other statements
```

User input for number of rows

```
rows = int(input("Enter the rows:"))
# Outer loop will print number of rows
for i in range(0,rows+1):
# Inner loop will print number of Astrisk
    for j in range(i):
        print("*",end = '')
    print()
```

```
Enter the rows:5
*
**
***
****
 *****
```

While loop

- **`while` loop**: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)

- Syntax:

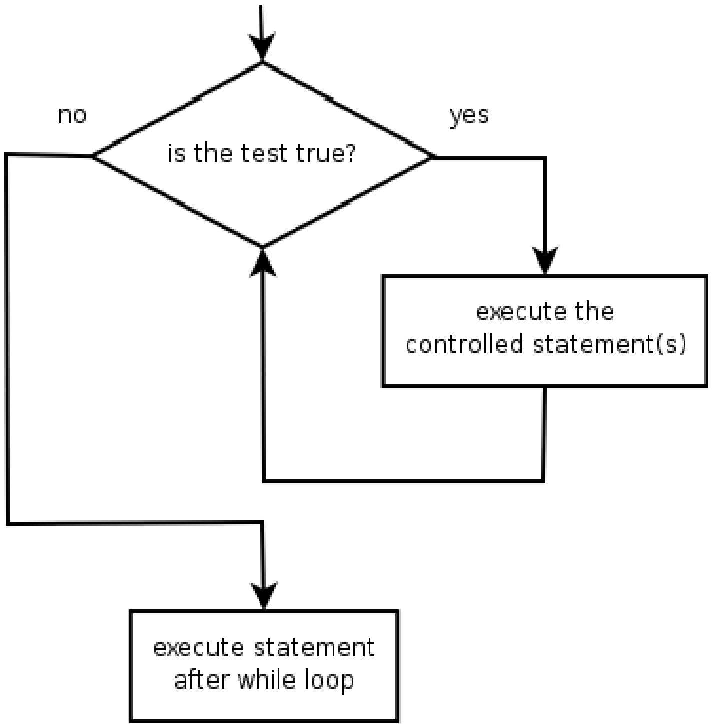
```
while condition:
    statements
```

- Example:

```
number = 1
while number < 200:
    print number,
    number = number * 2
```

  - Output:

```
1 2 4 8 16 32 64 128
```



no
is the test true?
yes
execute the controlled statement(s)
execute statement after while loop

Python Arrays

Arrays are used to store multiple values in one single variable.The Array can be created in Python by importing the array module to the python program.

```
from array import *
arrayName = array(typecode, [initializers])
```

Accessing array elements

We can access the array elements using the respective indices of those elements.

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print("First element:", a[0])
print("Second element:", a[1])
print("Second last element:", a[-1])
```

Output:

```
First element: 2
 Second element: 4
 Second last element: 8
```

Change or Add elements

Arrays are mutable, and their elements can be changed in a similar way like lists.

```
import array as arr
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])
# changing first element
numbers[0] = 0
print(numbers)    # Output: array('i', [0, 2, 3, 5, 7, 10])

# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)    # Output: array('i', [0, 2, 4, 6, 8, 10])
```

The Length of an Array

- Use the len() method to return the length of an array.

Example : Return the number of elements in the cars array:

```
cars = ["Ford", "Volvo", "BMW"]
x = len(cars)
print(x)
```

Output:

```
3
```

Array Concatenation

We can easily concatenate any two arrays using the + symbol.

```
a=arr.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])
b=arr.array('d',[3.7,8.6])
c=arr.array('d')
c=a+b
print("Array c = ",c)
```

Output:

```
Array c= array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7,8.6])
```

Array Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python String

We can create a string by enclosing the characters in single-quotes or double-quotes.

Python also provides triple-quotes to represent the string, but it is generally used for multiline string

Using single quotes

str1 = 'Hello Python'

print(str1)

Using double quotes

str2 = "Hello Python"

print(str2)

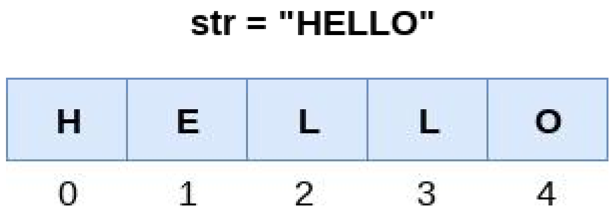
Using triple quotes

str3 = '''''Triple quotes are generally used for
  represent the multiline or
  docstring'''

print(str3)

Strings indexing and splitting

Like other languages, the indexing of the Python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.



str[0] = 'H'

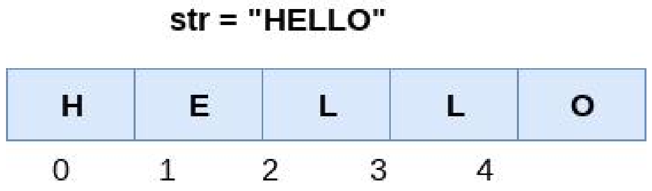
str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

- the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in Python to access the substring from the given string. Consider the following example.



str[0] = 'H'	str[:] = 'HELLO'
str[1] = 'E'	str[0:] = 'HELLO'
str[2] = 'L'	str[:5] = 'HELLO'
str[3] = 'L'	str[:3] = 'HEL'
str[4] = 'O'	str[0:2] = 'HE'
	str[1:4] = 'ELL'

- Deleting the String
- As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword**.**

```
str = "JAVATPOINT"
del str[1]
```

Output:

```
TypeError: 'str' object doesn't support item deletion
```

```
str1 = "JAVATPOINT"
del str1
print(str1)
```

Output:

```
NameError: name 'str1' is not defined
```

- Reassigning Strings
- Updating the content of the strings is as easy as assigning it to a new string.
- A string can only be replaced with new string since its content cannot be partially replaced.
- str = "HELLO"
- str[0] = "h"
- **print**(str) not possible

- str = "HELLO"
- **print**(str)
- str = "hello"
- **print**(str)
- output
- HELLO
- hello

String Operators

+	**It is known as concatenation operator used to join the strings given either side of the operator.**
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

Python Functions

A function is a block of code which only runs when it is called.You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

Example

```
def my_function():
    print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
# function definition
def hello_world():
   print("hello world")
# function calling
hello_world()
```

Output:

```
hello world
```

The return statement

The return statement is used at the end of the function and returns the result of the function.

```
# Defining function
def sum():
   a = 10
   b = 20
   c = a+b
   return c
# calling sum() function in print statement
print("The sum is:",sum())
```

Arguments in python

- There may be several types of arguments which can be passed at the time of function call.
- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

• Arguments in function

- The arguments are types of information which can be passed into the function.
- #defining the function
- **def** func (name):
-     **print**("Hi ",name)
- #calling the function
- func("Devansh")

Required Arguments

The required arguments are concerned, these are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition

```
def func(name):
    message = "Hi "+name
    return message
name = input("Enter the name:")
print(func(name))
```

Default Arguments

- Python allows us to initialize the arguments at the function definition.

```
def printme(name,age=22):
 print("My name is",name,"and age is",age)
printme(name = "john")
```

Keyword Arguments

Python allows us to call the function with the keyword arguments. The name of the arguments is treated as the keywords and matched in the function calling and definition.

This way the order of the arguments does not matter.

Example

```
def simple_interest(p,t,r):
   return (p*t*r)/100
print("Simple Interest: ",simple_interest(t=10,r=10,p=1900))
```

Variable-length Arguments (*args)

- By using the variable-length arguments, we can pass any number of arguments.
- However, at the function definition, we define the variable-length argument using the ***args** (star) as *<variable - name >
- **def** printme(*names):
-     **print**("type of passed argument is ",type(names))
-     **print**("printing the passed arguments...")
-     **for** name **in** names:
-         **print**(name)
- printme("john","David","smith","nick")

Call by reference in Python

- In Python, call by reference means passing the actual value as an argument in the function. All the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.

- #defining the function
- **def** change_list(list1):
-     list1.append(20)
-     list1.append(30)
-     **print**("list inside function = ",list1)
-
- #defining the list
- list1 = [10,30,40,50]
-
- #calling the function
- change_list(list1)
- **print**("list outside function = ",list1)

THANK YOU