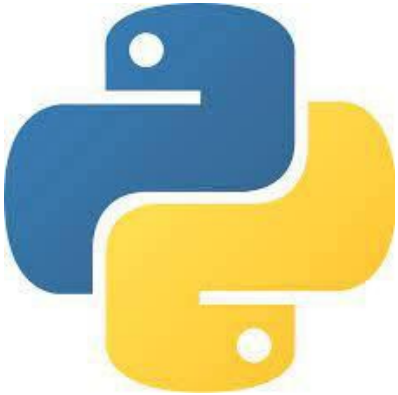
python Programming

Prasad Banoth

K4U Infotech
(Knowledge for U)


Guido
van
Rossum


MONTY PYTHON'S
FLYING
CIRCUS



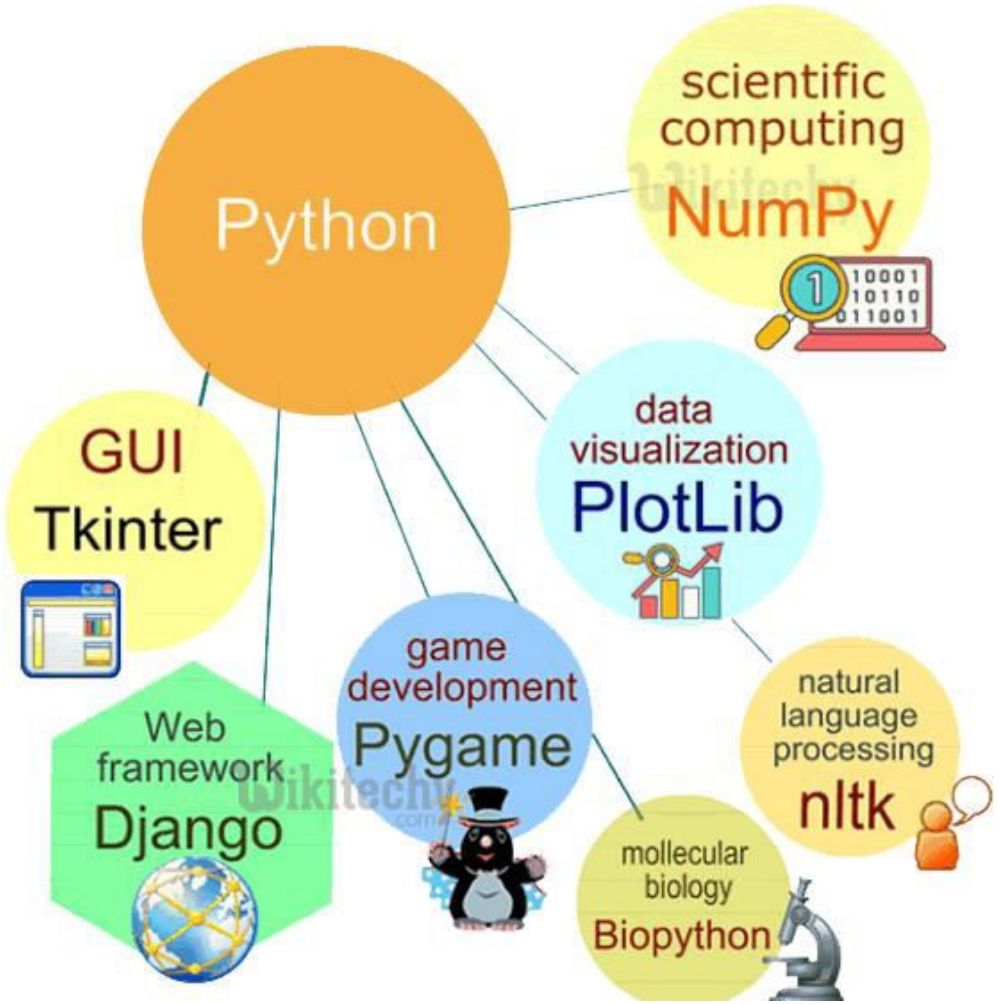
Using C programming

```c
// C program to add two numbers
#include<stdio.h>

int main()
{
   int A,  B,  sum  =  0;

   // Ask user to enter the two numbers
   printf("Enter two numbers A and B : \n");

   // Read two numbers from the user || A = 5, B = 7
   scanf("%d%d", &A, &B);

   // Calclulate the addition of A and B
   // using '+' operator
   sum = A + B;

   // Print the sum
   printf("Sum of A and B is: %d", sum);

   return 0;
}
```

Using python

Python used by

python Applications

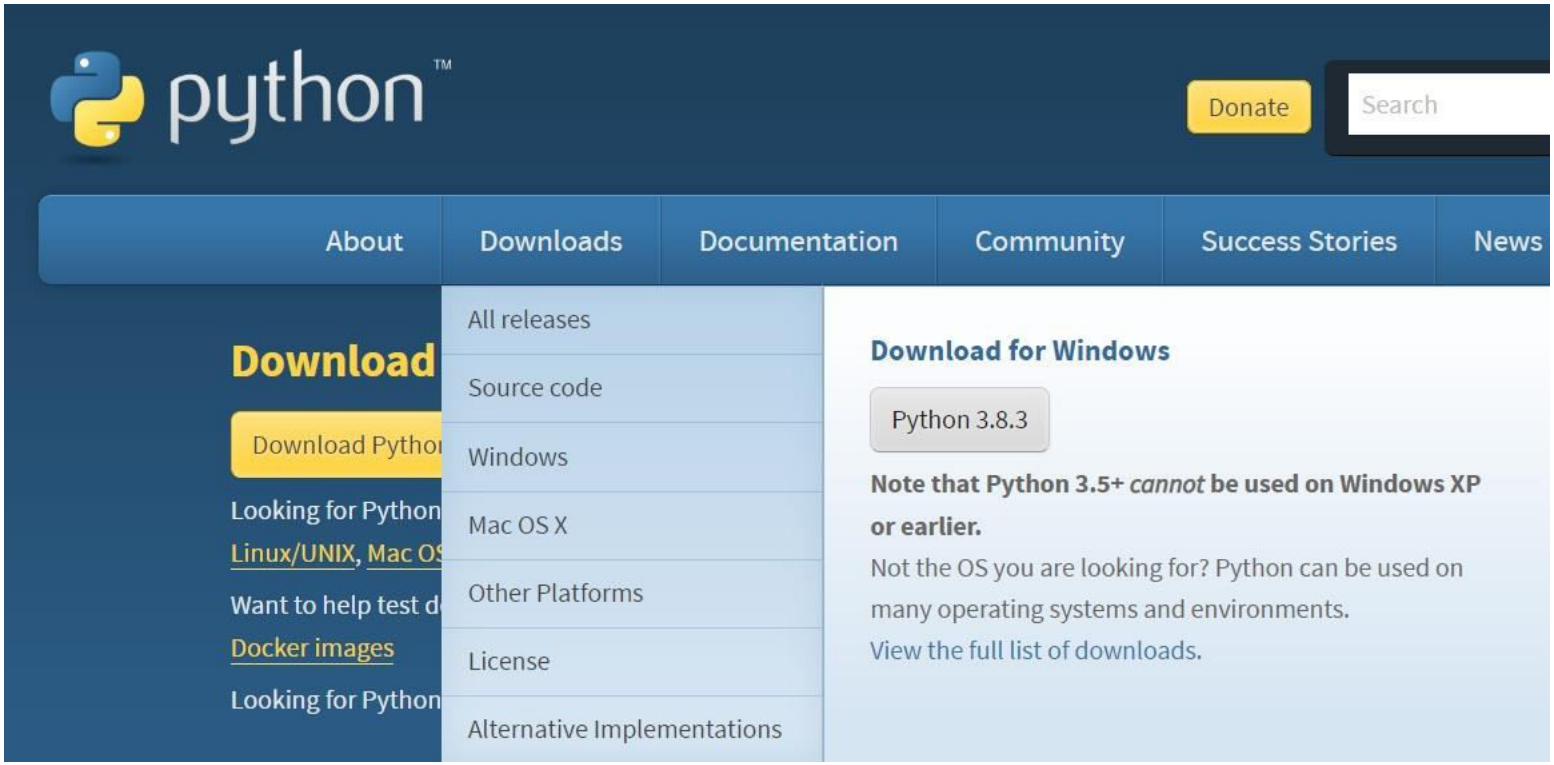


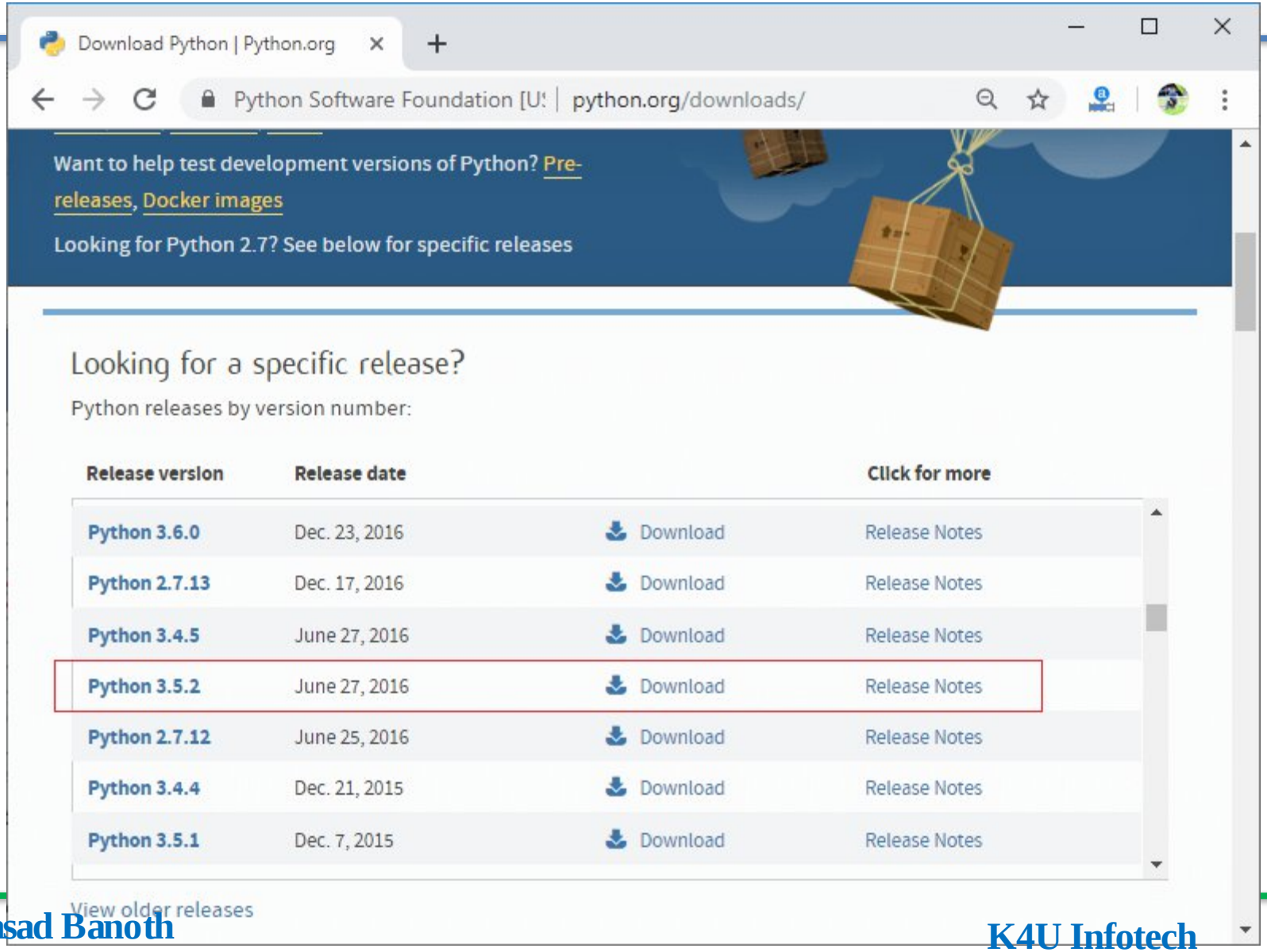
Tools for Implementing python code

Installing python Software

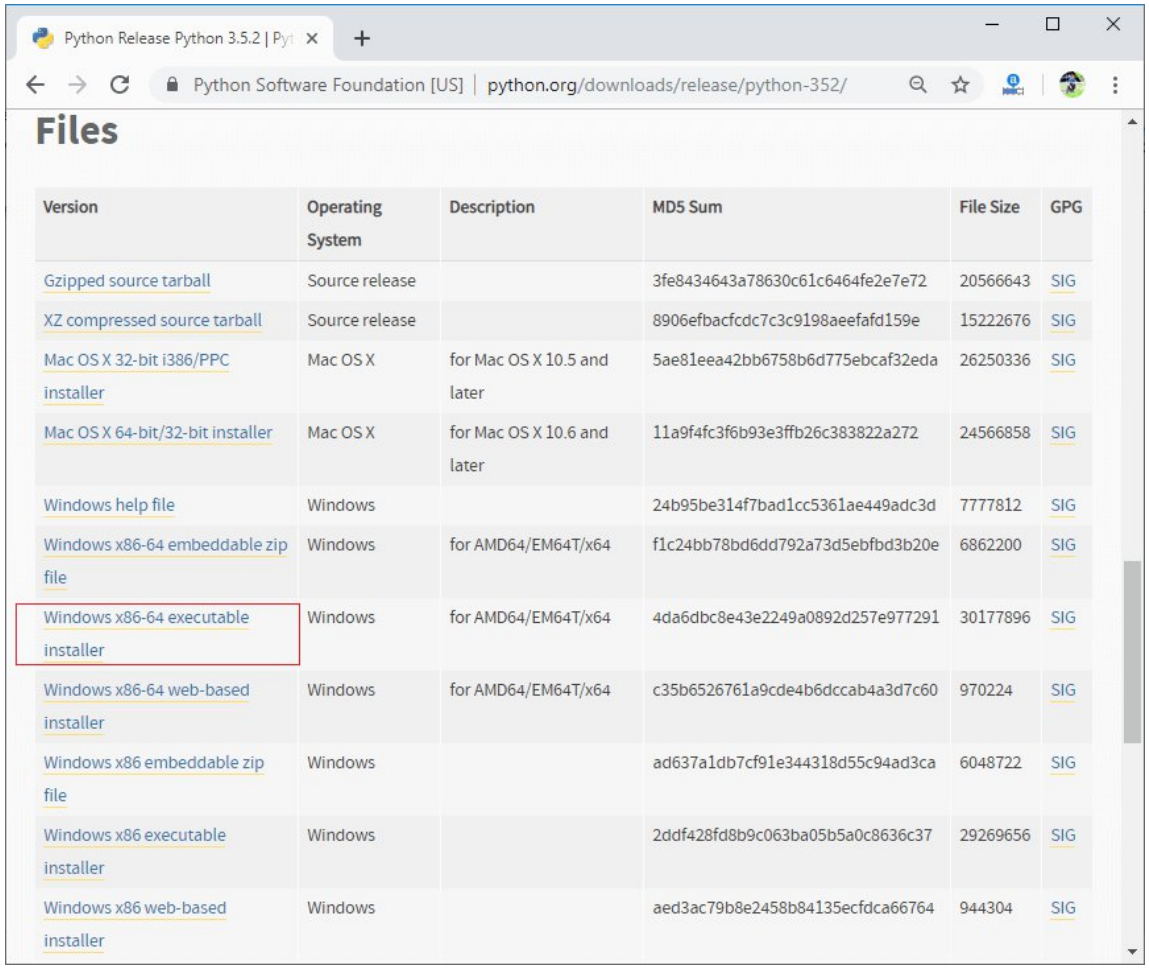
Download Python from the below link:

https://www.python.org/downloads/



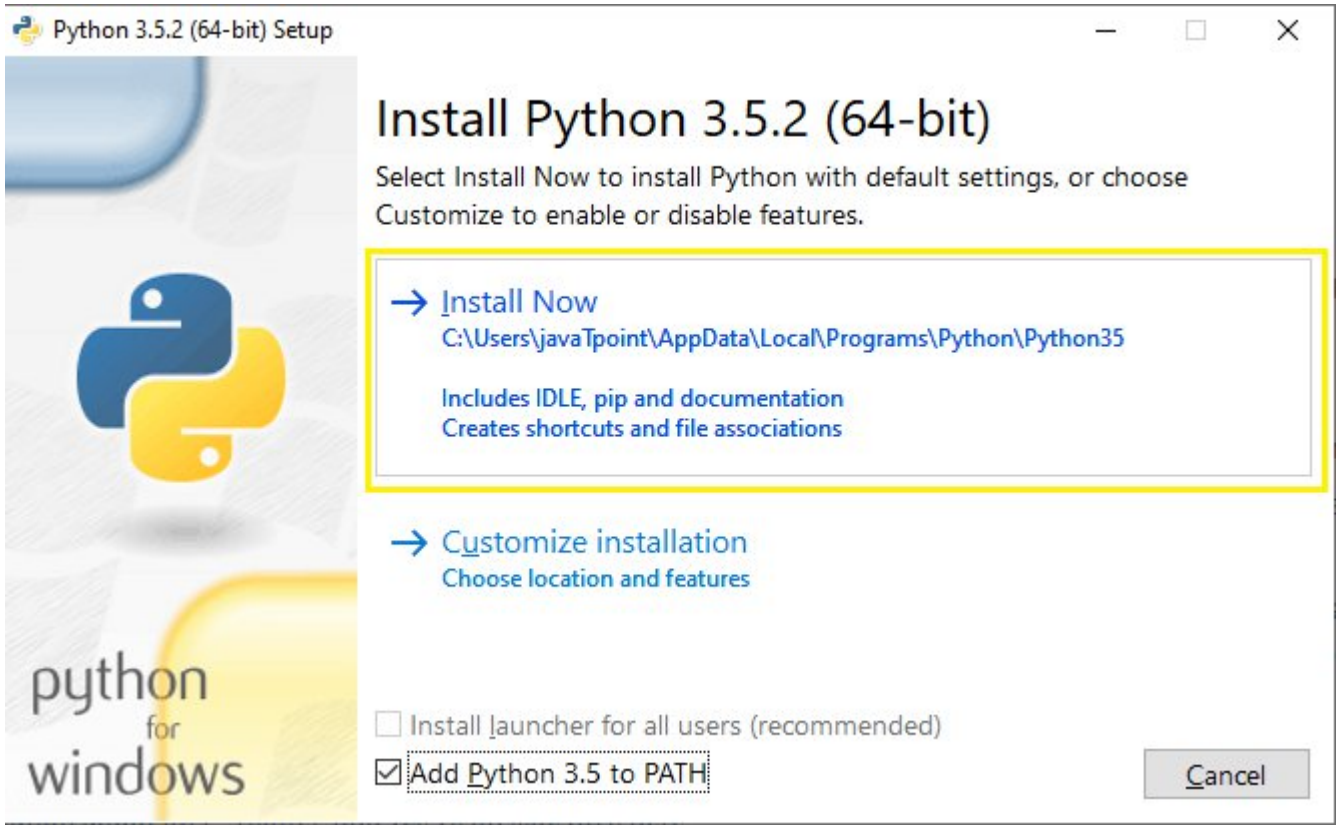

Download Python | Python.org
Python Software Foundation [US]
python.org/downloads/
Want to help test development versions of Python? Pre-releases, Docker images
Looking for Python 2.7? See below for specific releases
Looking for a specific release?
Python releases by version number:
Release version
Release date
Click for more
Python 3.6.0 Dec. 23, 2016 Download Release Notes
Python 2.7.13 Dec. 17, 2016 Download Release Notes
Python 3.4.5 June 27, 2016 Download Release Notes
Python 3.5.2 June 27, 2016 Download Release Notes
Python 2.7.12 June 25, 2016 Download Release Notes
Python 3.4.4 Dec. 21, 2015 Download Release Notes
Python 3.5.1 Dec. 7, 2015 Download Release Notes
View older releases

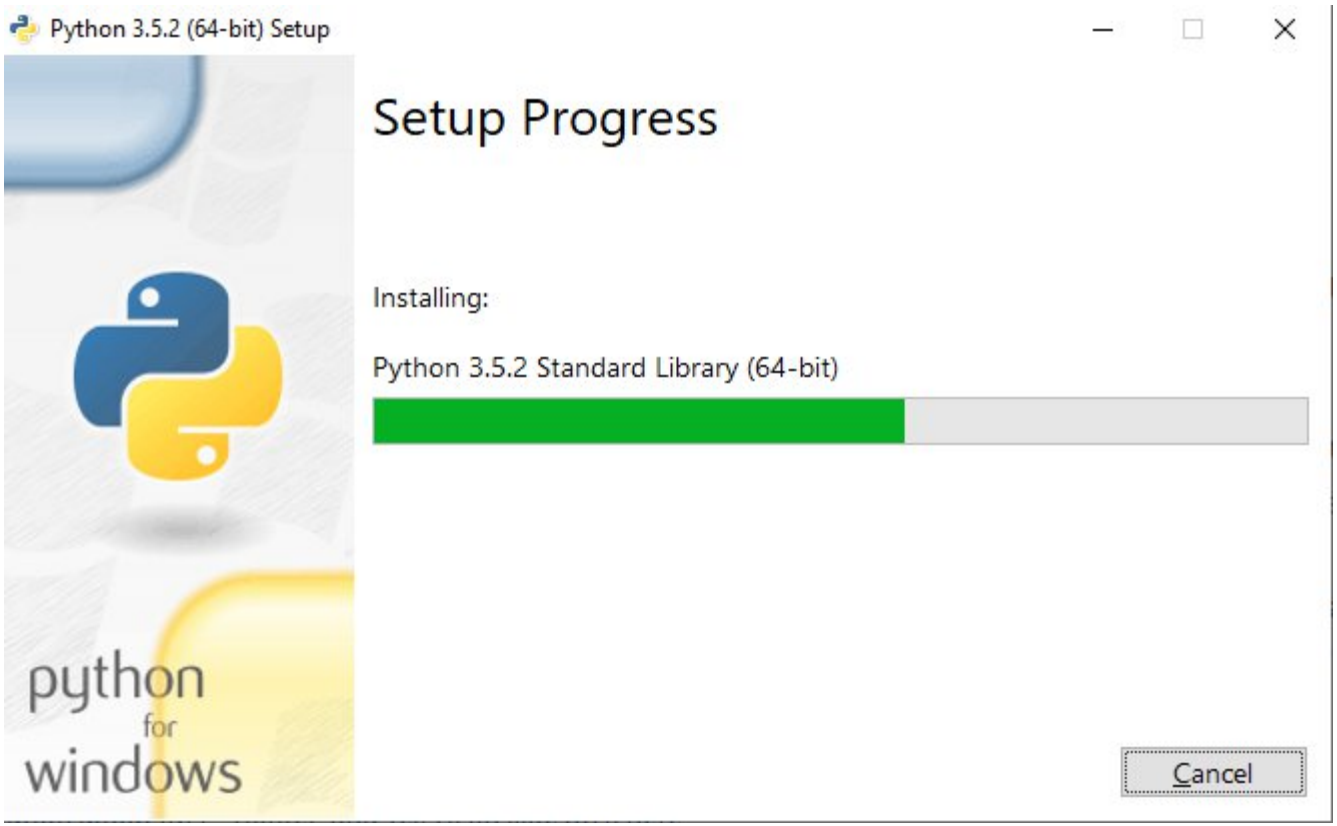
Step: We will be brought to another page, where we will need to select either the **x86-64** or **amd64 installer** to install Python.
We use here **Windows x86-64 executable installer**.

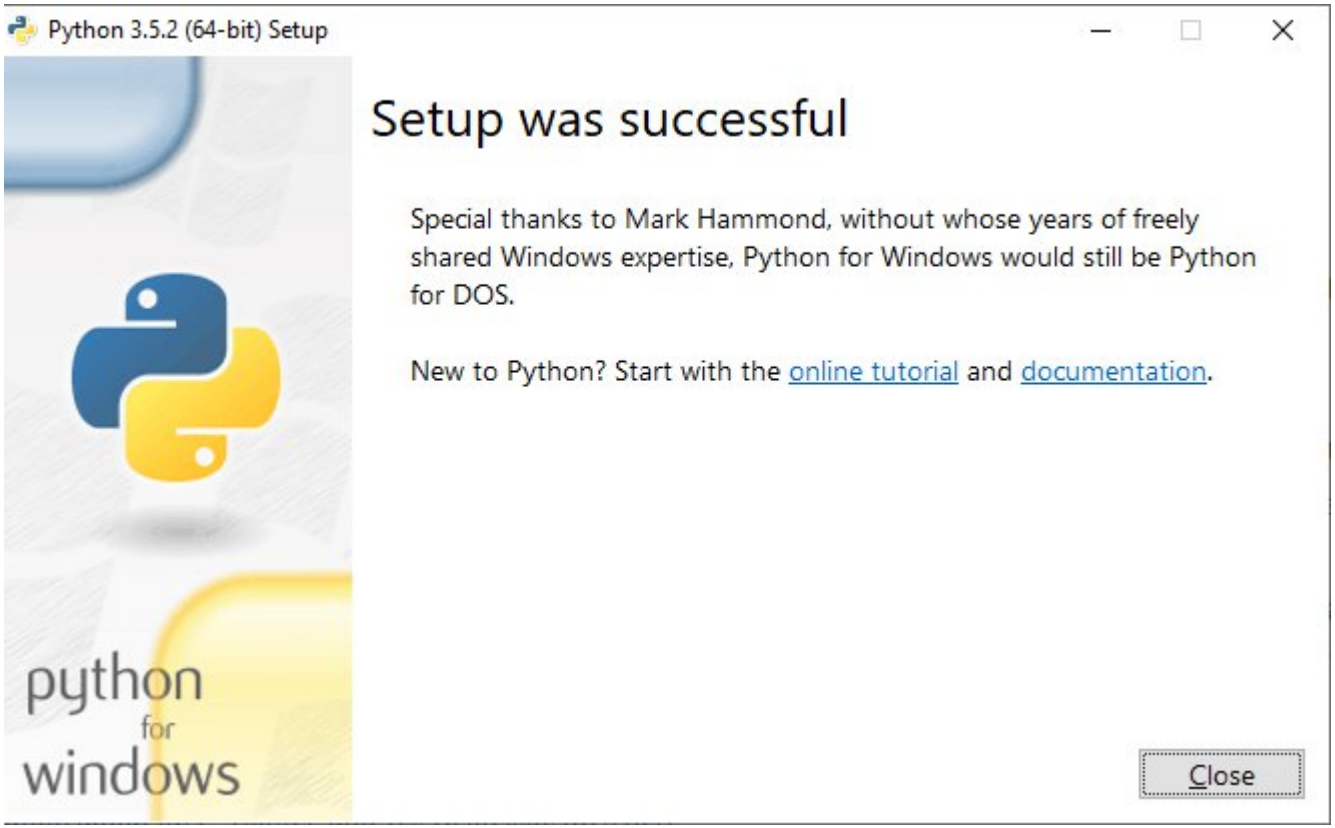


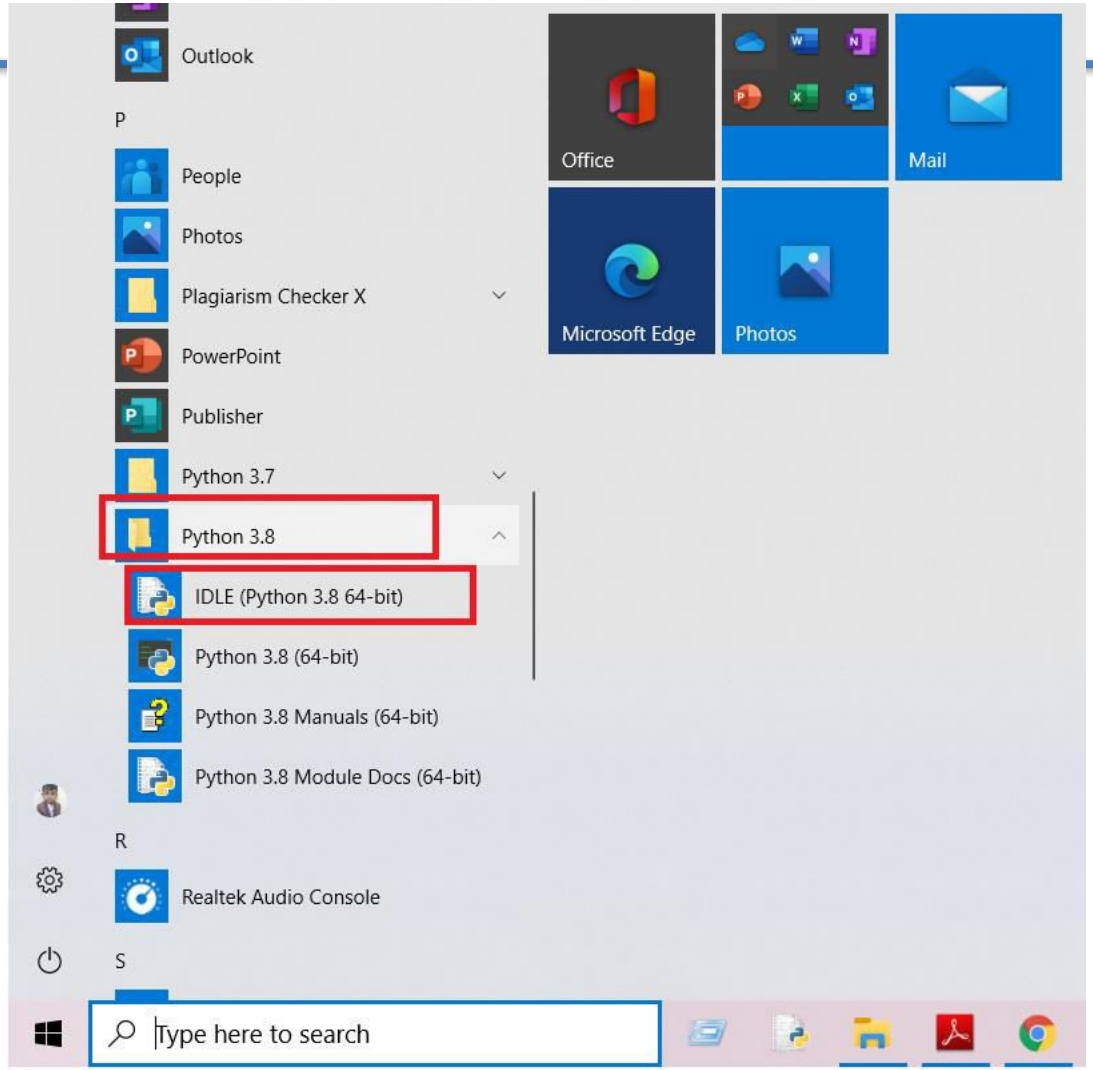
Files

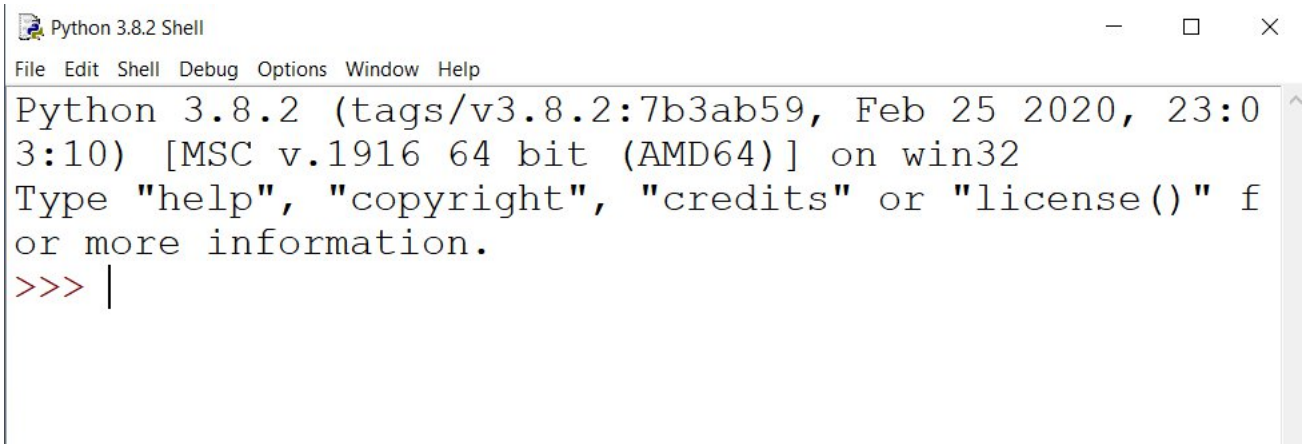
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		3fe8434643a78630c61c6464fe2e7e72	20566643	SIG
XZ compressed source tarball	Source release		8906efbacfcdc7c3c9198aeefafd159e	15222676	SIG
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	5ae81eea42bb6758b6d775ebcaf32eda	26250336	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	11a9f4fc3f6b93e3ffb26c383822a272	24566858	SIG
Windows help file	Windows		24b95be314f7bad1cc5361ae449adc3d	7777812	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	f1c24bb78bd6dd792a73d5ebfbd3b20e	6862200	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	4da6dbc8e43e2249a0892d257e977291	30177896	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	c35b6526761a9cde4b6dccab4a3d7c60	970224	SIG
Windows x86 embeddable zip file	Windows		ad637a1db7cf91e344318d55c94ad3ca	6048722	SIG
Windows x86 executable installer	Windows		2ddf428fd8b9c063ba05b5a0c8636c37	29269656	SIG
Windows x86 web-based installer	Windows		aed3ac79b8e2458b84135ecfdca66764	944304	SIG

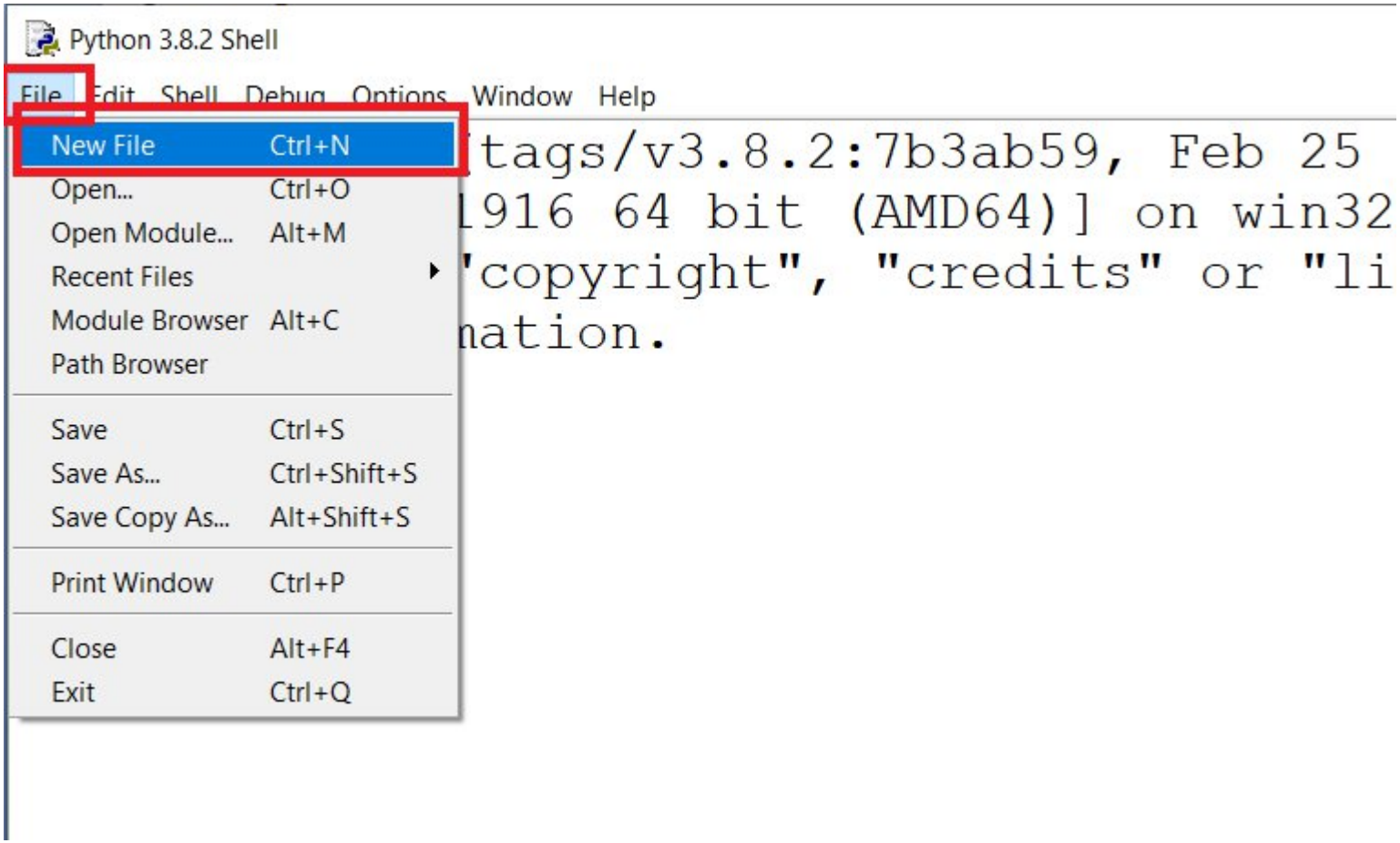

Python 3.5.2 (64-bit) Setup
Install Python 3.5.2 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
Install Now
C:\Users\javaTpoint\AppData\Local\Programs\Python\Python35
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
python
for
windows
Install launcher for all users (recommended)
Add Python 3.5 to PATH
Cancel


Python 3.5.2 (64-bit) Setup
Setup Progress
Installing:
Python 3.5.2 Standard Library (64-bit)
python
for
windows
Cancel

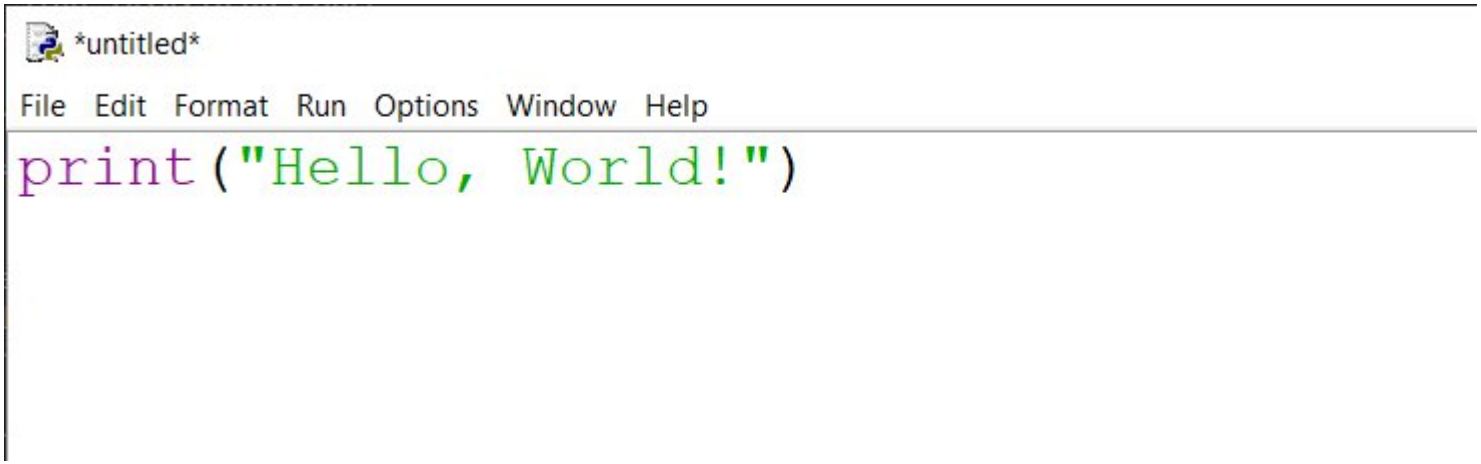

Python 3.5.2 (64-bit) Setup
Setup was successful
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.
New to Python? Start with the online tutorial and documentation.
python
for
windows
Close

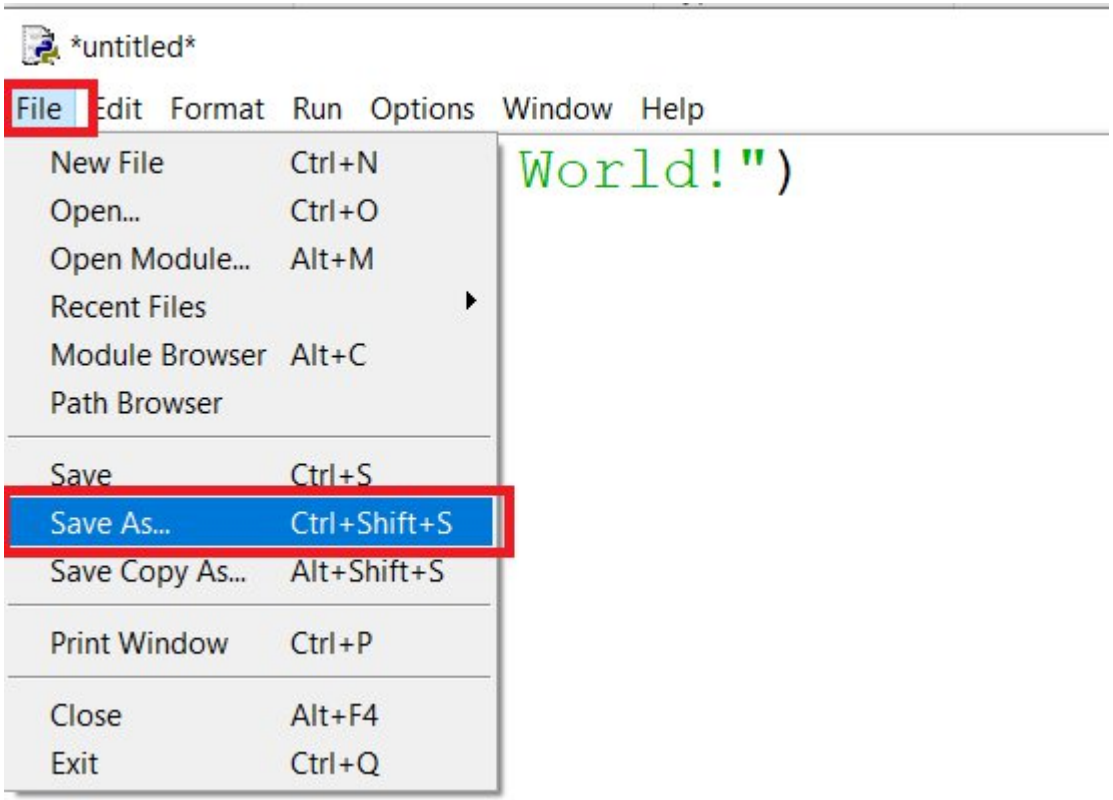

Outlook
P
People
Photos
Plagiarism Checker X
PowerPoint
Publisher
Python 3.7
Python 3.8
IDLE (Python 3.8 64-bit)
Python 3.8 (64-bit)
Python 3.8 Manuals (64-bit)
Python 3.8 Module Docs (64-bit)
R
Realtek Audio Console
S
Office
Mail
Microsoft Edge
Photos
Type here to search


Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:0
3:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" f
or more information.
>>>


Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
New File Ctrl+N
Open... Ctrl+O
Open Module... Alt+M
Recent Files
Module Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q
tags/v3.8.2:7b3ab59, Feb 25
1916 64 bit (AMD64)] on win32
'copyright", "credits" or "li
nation.


untitled
File Edit Format Run Options Window Help

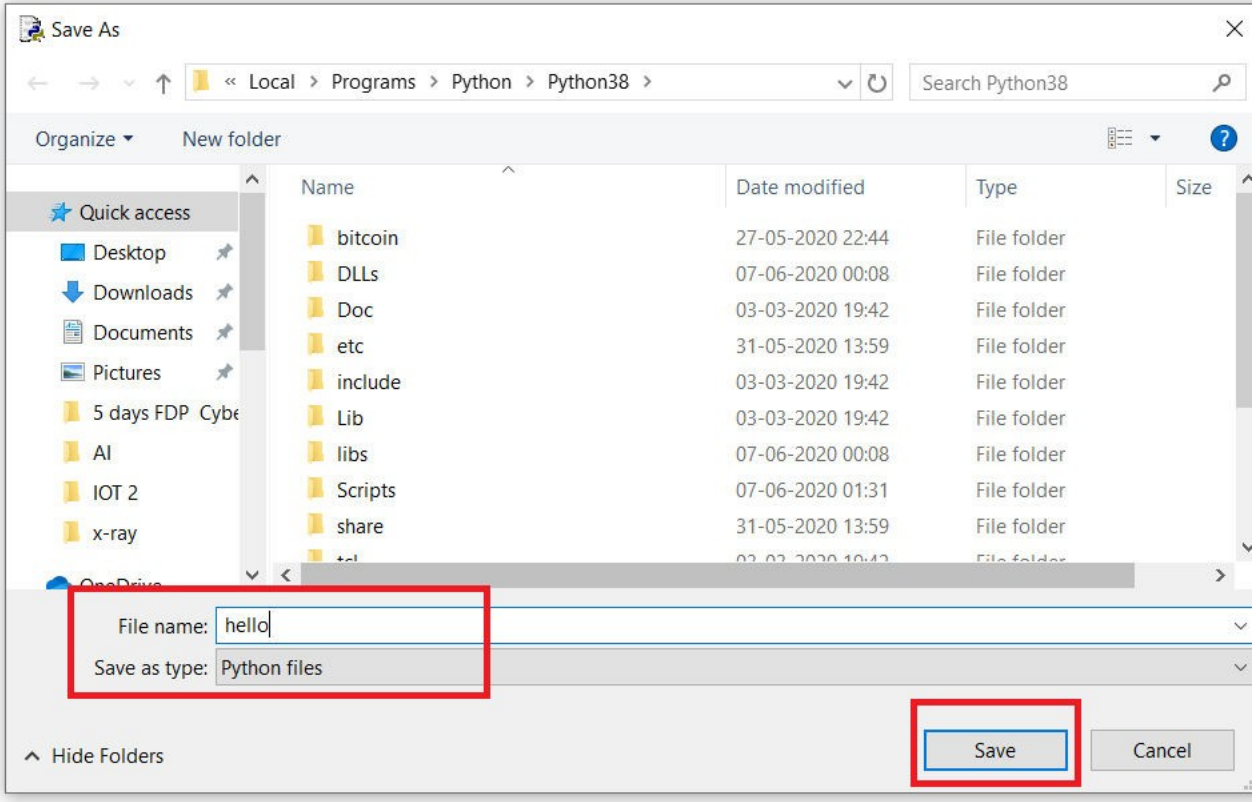

untitled
File Edit Format Run Options Window Help
print("Hello, World!")

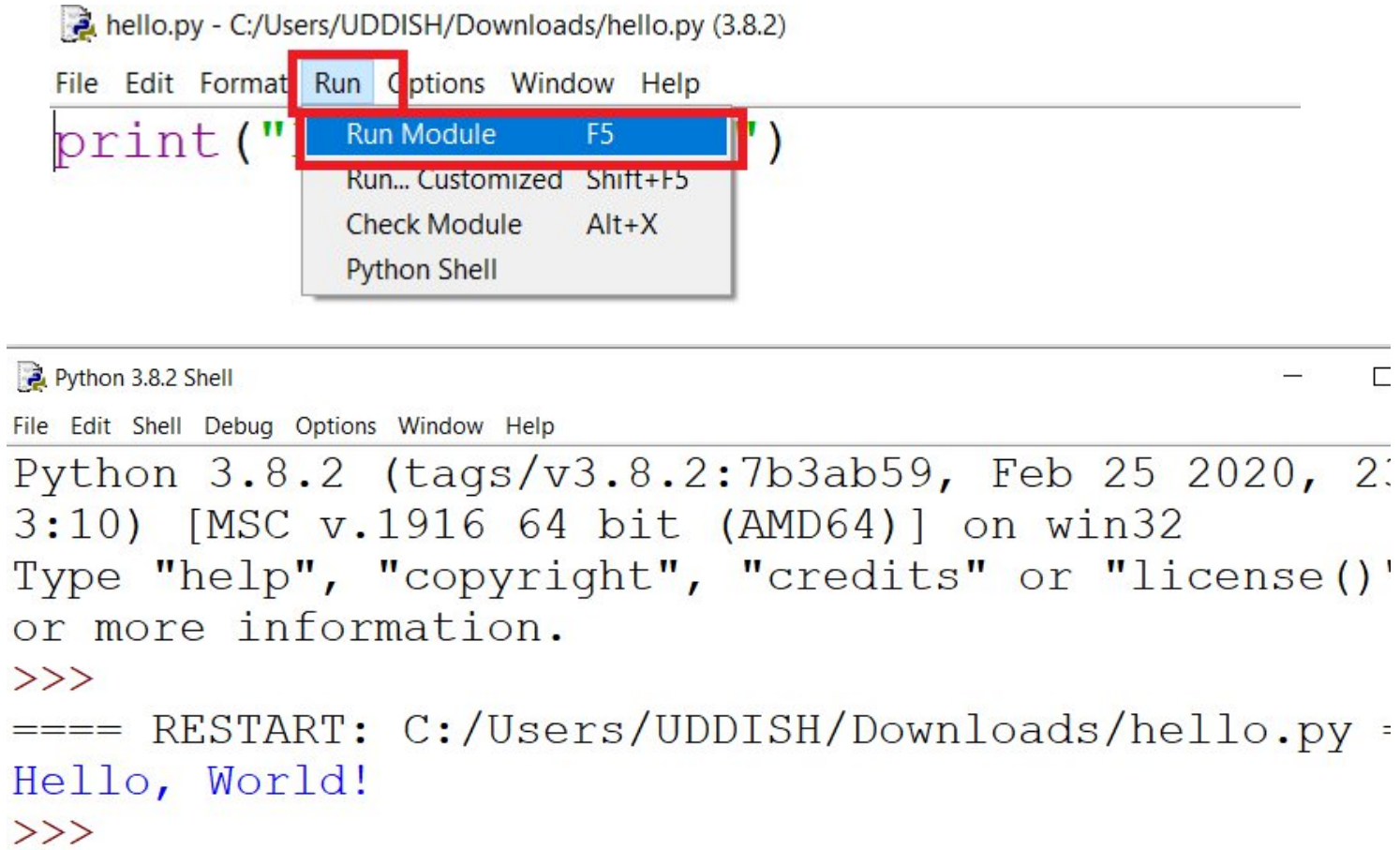

untitled
File Edit Format Run Options Window Help
World!")
New File Ctrl+N
Open... Ctrl+O
Open Module... Alt+M
Recent Files
Module Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q

untitled

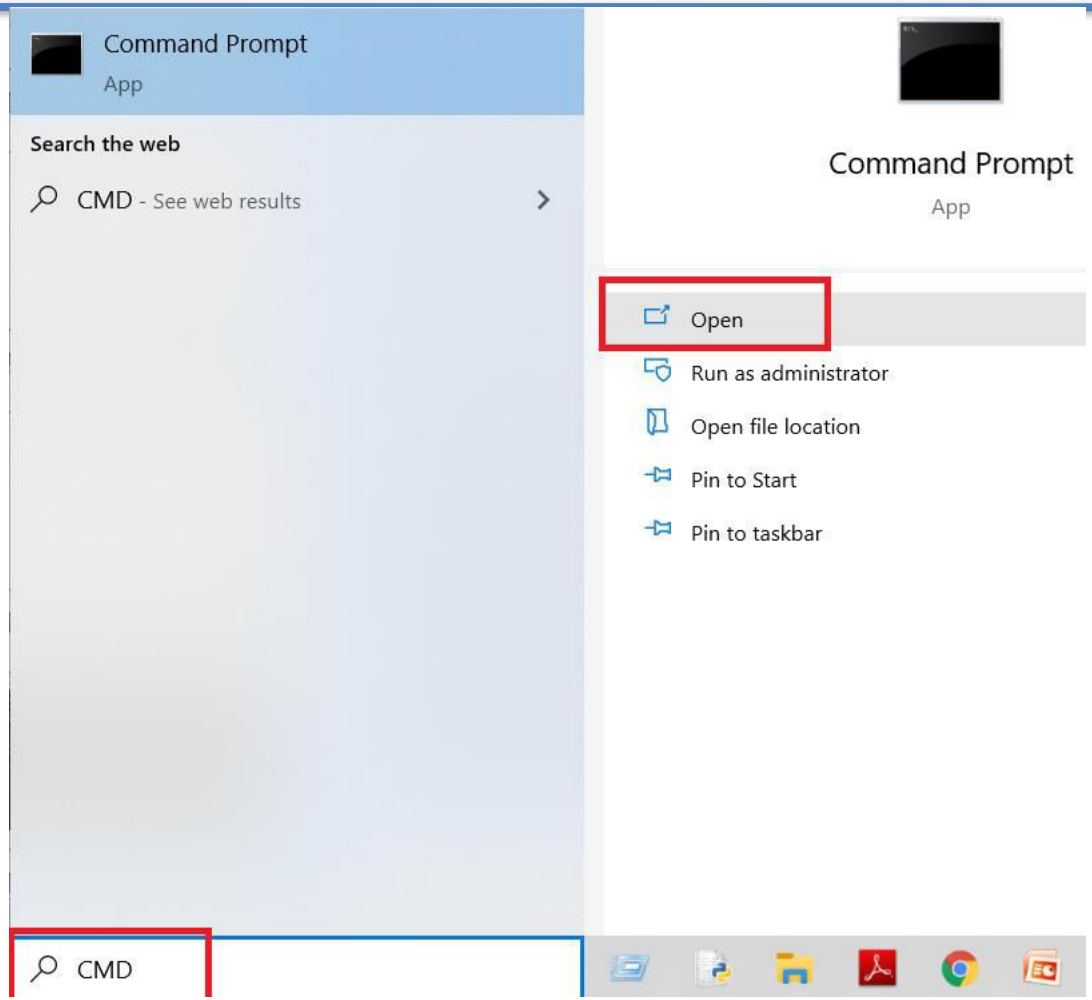
File Edit Format Run Options Window Help

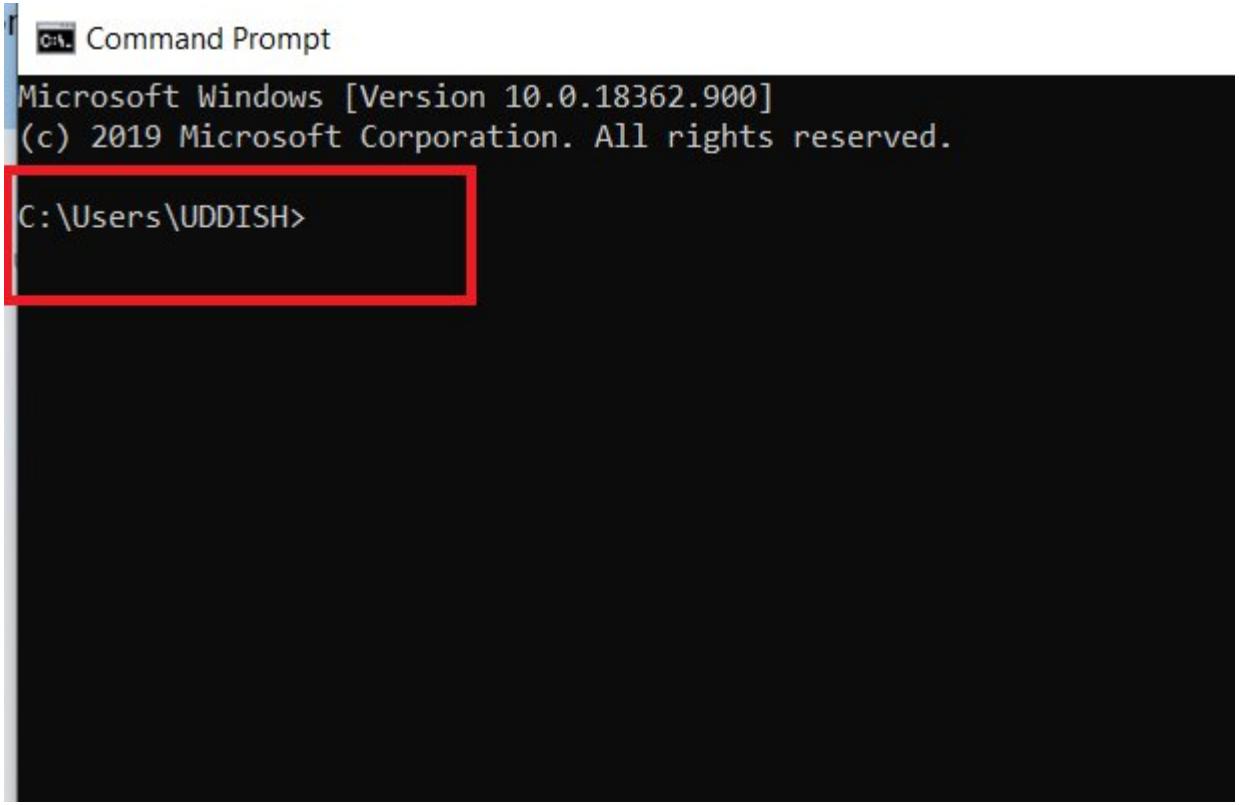
```
print("Hello, World!")
```

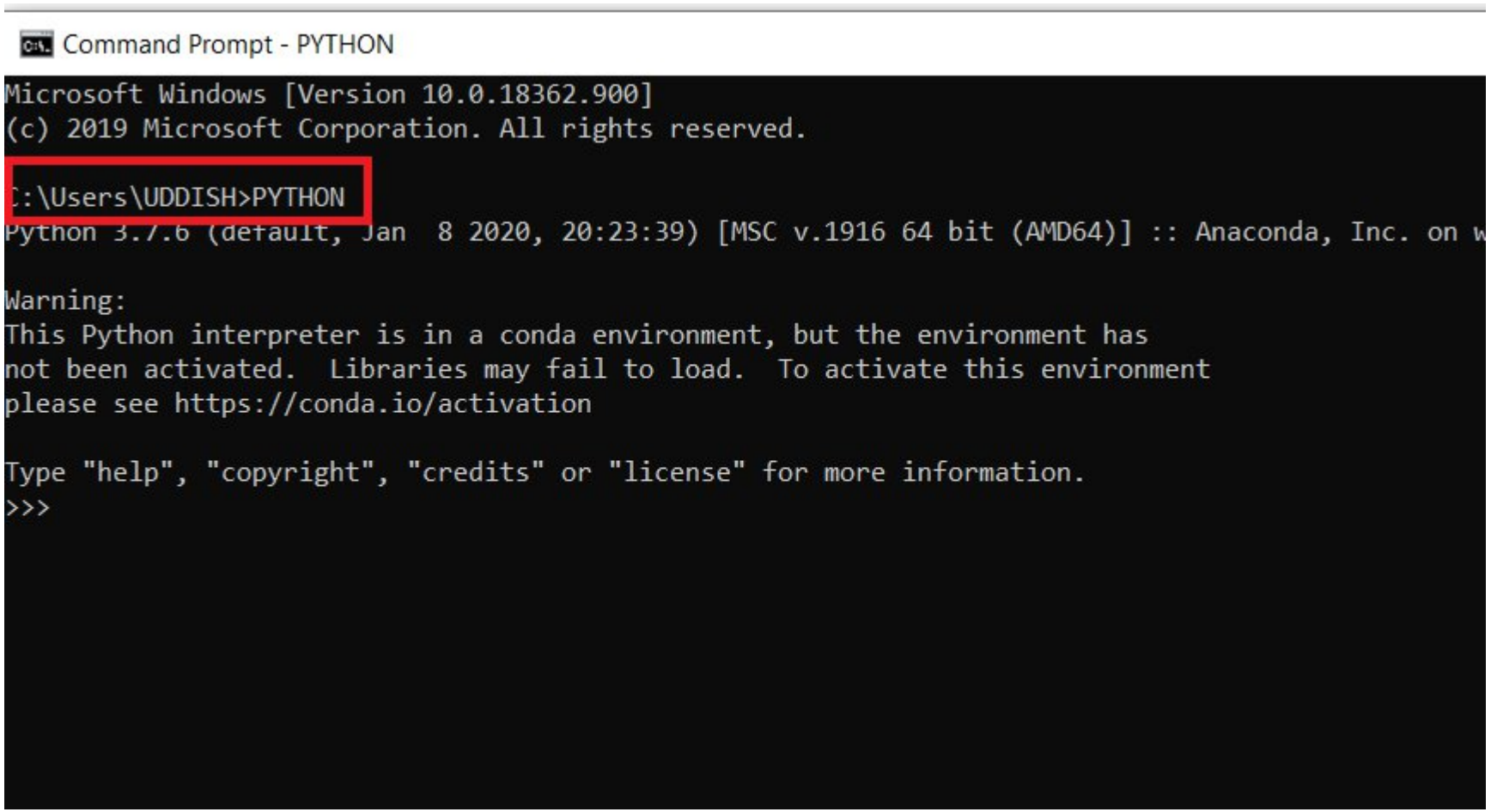


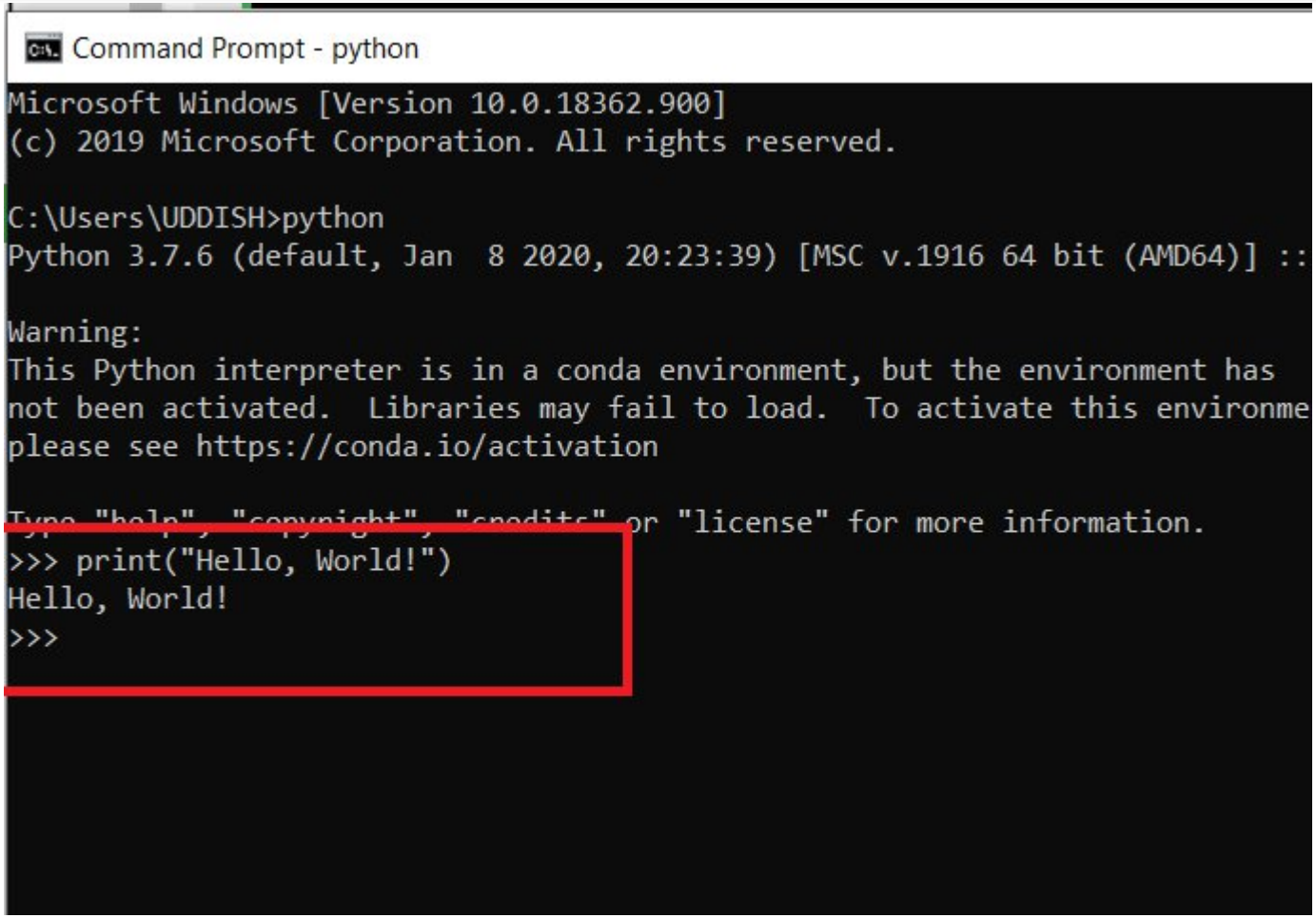

hello.py - C:/Users/UDDISH/Downloads/hello.py (3.8.2)
File Edit Format Run Options Window Help
print("
")
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 2
3:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()"
or more information.
>>>
==== RESTART: C:/Users/UDDISH/Downloads/hello.py =
Hello, World!
>>>

Check python version


Command Prompt
App
Search the web
CMD - See web results
Command Prompt
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
CMD


Command Prompt
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\UDDISH>


Command Prompt - PYTHON
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\UDDISH>PYTHON
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on w
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation
Type "help", "copyright", "credits" or "license" for more information.
>>>


Command Prompt - python
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\UDDISH>python
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] ::
Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environme
please see https://conda.io/activation
or "license" for more information.
>>> print("Hello, World!")
Hello, World!
>>>

Python Introduction

Prasad Banoth

K4U Infotech
(Knowledge for U)

What is Python?

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
- It is used for:
  - web development (server-side),
  - software development,
  - mathematics,
  - system scripting.

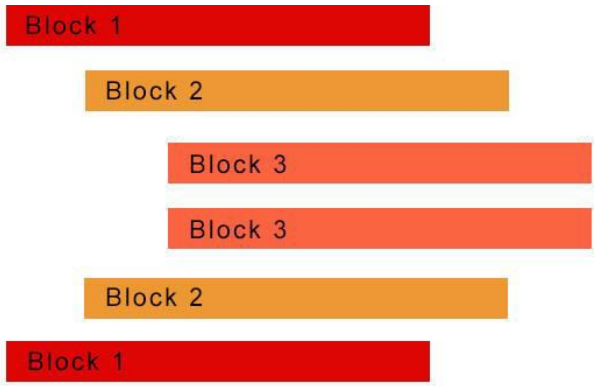
What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Python Indentation



Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

Python Indentation (cont..)

Example 1:

hello.py - C:/Users/UDDISH/Downloads/hello.py (3.8.2)

File Edit Format Run Options Window Help

```
if 5 > 2:
  print("Five is greater than two!")
```

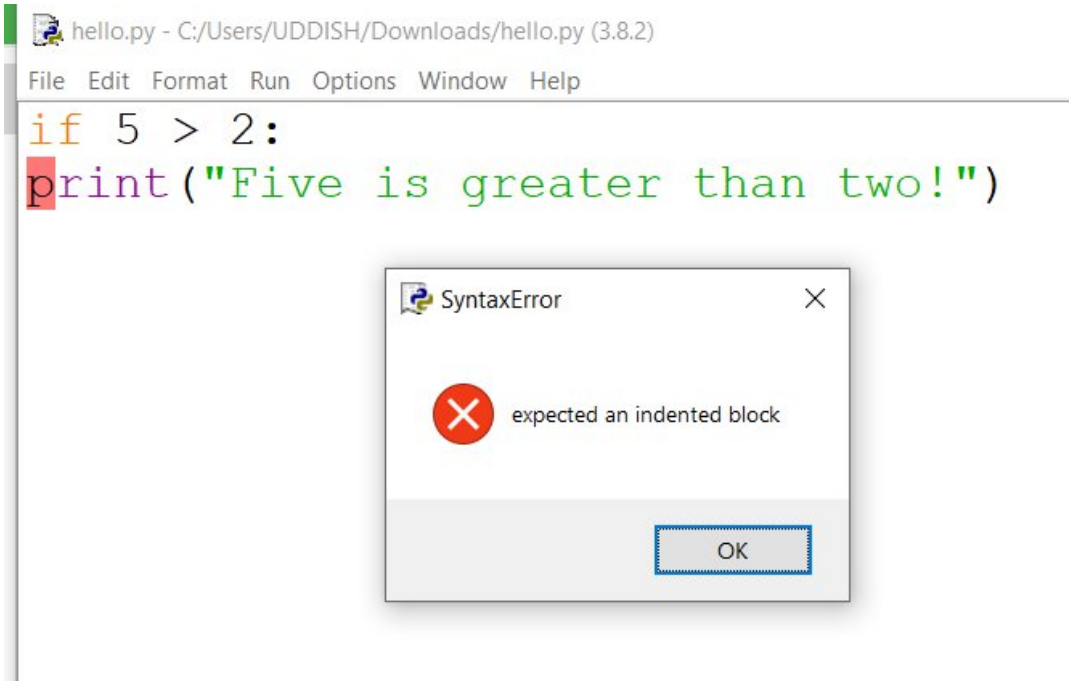
Output :

```
======
Five is greater than two!
>>>
```

Python Indentation (cont..)

- **Python will give you an error if you skip the indentation:**

Example 2:



The number of spaces is up to you as a programmer, but it has to be at **least one.**

Example 3:

```
if 5 > 2:
    print("Five is greater than two!")
if 5 > 2:
    print("Five is greater than two!")
```

```
if 5 > 2:
    print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example 4:

```
if 5 > 2:
    print("Five is greater than two!")
        print("Five is greater than two!")
```

Python Comments

Python Comments

- Comments can be used to
  - explain Python code.
  - make the code more readable.
  - prevent execution when testing code.

Creating a Comment

Comments starts with a #, and Python will ignore them:

Example 1:

```
#This is a comment
print("Hello, World!")
```

- Comments can be placed at the **end of a line,** and Python will ignore the rest of the line:

Example 2:

```
print("Hello, World!") #This is a comment
```

- Comments **does not have to be text** to explain the code, it can also be used to prevent Python from executing code:

Example 3:

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multi Line Comments

Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

Example 4:

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

Python Variables

Creating Variables:

- Variables are containers for storing data values.
- Unlike other programming languages, Python has no command for declaring a variable.

- A variable is created the moment you first assign a value to it.

Example 1:

```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

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

Example 2:

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

```
4
Sally
```

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

Example 3:

```
x = "Welcome"
print(x)
#double quotes are the same as single quotes:
x = 'John'
print(x)
```

```
Welcome
John
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Assign Value to Multiple Variables

- Python allows to assign values to multiple variables in one line:

Example 4:

```
x, y, z = "Orange", "Banana", "Cherry"

print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

And you can assign the *same* value to multiple variables in one line:

Example 5:

```
x = y = z = "Orange"

print(x)
print(y)
print(z)
```

```
Orange
Orange
Orange
```

Output Variables

- The Python print statement is often used to output variables.
- To combine both text and a variable, Python uses the + character:

Example 6:

```
x = "awesome"
print("Python is " + x)
```

```
Python is awesome
```

- You can also use the + character to add a variable to another variable:

```
x = "Python is "
y = "awesome"
z = x + y
print(z)
```

```
Python is awesome
```

- For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

```
15
```

- If you try to combine a string and a number, Python will give you an error:

```
x = 5
y = "John"
print(x + y)
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Global Variables

- Variables that are created outside of a function (as in all of the examples above) are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

```
Python is awesome
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

```
Python is fantastic
Python is awesome
```

Built-in Data Types

Built-in Data Types

- In programming, data type is an important concept.
- Variables can store data of different types, and different types can do different things.

Text Type:	**str**
Numeric Types:	**int, float, complex**
Sequence Types:	**list, tuple, range**
Mapping Type:	**dict**
Set Types:	**set, frozenset**
Boolean Type:	**bool**
Binary Types:	**bytes, bytearray, memoryview**

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Getting the Data Type

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

```
x = 5
print(type(x))
```

```
<class 'int'>
```

Setting the Data Type

```
x = "Hello World"

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
Hello World
<class 'str'>
```

```
x = 20.5

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
20.5
<class 'float'>
```

```
x = {"name" : "John", "age" : 36}

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
{'name': 'John', 'age': 36}
<class 'dict'>
```

Setting the Specific Data Type

```
x = str("Hello World")

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
Hello World
<class 'str'>
```

```
x = float(20.5)

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
20.5
<class 'float'>
```

```
x = dict(name="John", age=36)

#display x:
print(x)

#display the data type of x:
print(type(x))
```

```
{'name': 'John', 'age': 36}
<class 'dict'>
```

Python Numbers

Python Numbers

There are three numeric types in Python:

- **Int :** Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length. (x = 1)
- **Float :** Float, or "floating point number" is a number, positive or negative, containing one or more decimals. (y = 2.8)
- **Complex :** Complex numbers are written with a "j" as the imaginary part: (z = 1j)

Type Conversion

- You can convert from one type to another with the int(), float(), and complex() methods:

```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(x)

print(x)
print(y)
print(z)
```

```
1.0
2
(1+0j)
<class 'float'> <class 'int'> <class 'complex'>
```

Note: You cannot convert complex numbers into another number type.

Random Number

- Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

```
import random

print(random.randrange(1, 10))
```

```
6
```

Python Strings

String Literals

- String literals in python are surrounded by either single quotation marks, or double quotation marks.

ex: 'hello' is the same as "hello".

You can display a string literal with the print() function:

Python Collections (Arrays)

- List
- Tuple
- Set
- Dictionary

There are **four collection data types** in the Python programming language:

- **List** is a collection which is **ordered and changeable.** Allows duplicate members.
- **Tuple** is a collection which is **ordered and unchangeable.** Allows **duplicate members.**
- **Set** is a collection which is **unordered and unindexed**. **No duplicate members.**
- **Dictionary** is a collection which is **unordered, changeable and indexed. No duplicate members.**

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Python Collections - List

Python - List

mylist = ["apple", "banana", "cherry"]

1. Lists are used to store multiple items in a **single variable.**
2. **List** is a collection which is **ordered and changeable.**
3. Allows **duplicate members.**
4. In Python lists are written with **square brackets.**

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

List Items

- List items are **ordered, changeable, and allow duplicate values.**
- List items are indexed, the first item has index [0], the second item has index [1]

  etc.

List is a collection which is **ordered and changeable.** Allows **duplicate members**

Ordered : When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

Changeable : The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates : Since lists are indexed, lists can have items with the same value

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

List Length

- To determine how many items a list has, use the len() function:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
3
```

List Items - Data Types

- List items can be of any data type:

Example : String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

print(list1)
print(list2)
print(list3)
```

```
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
```

Example : A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]

print(list1)
```

```
['abc', 34, True, 40, 'male']
```

type() ?

type()

- From Python's perspective, lists are defined as objects with the data type 'list':

<class 'list'>

Example : What is the data type of a list?

```
mylist = ["apple", "banana", "cherry"]

print(type(mylist))
```

```
<class 'list'>
```

Python - Access List Items

Access Items :

- List items are indexed and you can access them by referring to the index number:

Example : Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
banana
```

Note: The first item has index 0.

Negative Indexing

- Negative indexing means start from the end
- -1 refers to the last item, -2 refers to the second last item etc.

Example : Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

```
cherry
```

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

Example : Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])

#This will return the items from position 2 to 5.

#Remember that the first item is position 0,
#and note that the item in position 5 is NOT included
```

```
['cherry', 'orange', 'kiwi']
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

Example: This example returns the items from the beginning to, but NOT including, "kiwi":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

```
['apple', 'banana', 'cherry', 'orange']
```

Example : This example returns the items from "cherry" to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

Range of Negative Indexes

- Specify negative indexes if you want to start the search from the end of the list:

Example : This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

```
['orange', 'kiwi', 'melon']
```

Check if Item Exists

- To determine if a specified item is present in a list use the in keyword:

Example : Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

```
Yes, 'apple' is in the fruits list
```

Python - Change List Items

Change Item Value :

To change the value of a specific item, refer to the index number:

Example : Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"

print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

Change a Range of Item Values

- To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example : Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

If you insert ***more*** **items** than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example : Change the second value by replacing it with *two* new values:

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'cherry']
```

If you insert ***less*** **items** than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example : Change the second and third value by replacing it with *one* value:

```
thislist = ["apple", "banana", "cherry"]

thislist[1:3] = ["watermelon"]

print(thislist)
```

```
['apple', 'watermelon']
```

Insert Items :

- To insert a new list item, **without replacing any of the existing values,** we can use the insert() method.
- The insert() method inserts an item at the specified index:

Example : Insert "watermelon" as the third item:

```
thislist = ["apple", "banana", "cherry"]

thislist.insert(2, "watermelon")

print(thislist)
```

```
['apple', 'banana', 'watermelon', 'cherry']
```

Python - Add List Items

Append Items :

To add an item to the **end of the list**, use the append() method:

Example : Using the append() method to append an item:

```
thislist = ["apple", "banana", "cherry"]

thislist.append("orange")

print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

Insert Items:

- To insert a list item at a specified index, use the insert() method.
- The insert() method inserts an item at the specified index:

Example : Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

Extend List :

- To append elements from *another list* to the current list, use the extend() method.

Example : Add the elements of tropical to thislist:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]

thislist.extend(tropical)

print(thislist)
```

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
```

Python - Remove List Items

Remove Specified Item :

The remove() method removes the specified item.

Example : Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry']
```

Remove Specified Index :

- The pop() method removes the specified index.

Example : Remove the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

```
['apple', 'cherry']
```

- If you do not specify the index, the pop() method removes the last item.

Example : Remove the last item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

```
['apple', 'banana']
```

- The del keyword also removes the specified index:

Example : Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

- The del keyword can also delete the list completely.

Example : Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Clear the List :

- The clear() method empties the list.
- The list still remains, but it has no content.

Example : Clear the list content:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

List Methods

- Python has a set of built-in methods that you can use on lists.

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 item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python Collections - TUPLES

Prasad Banoth

K4U Infotech
(Knowledge for U)

Python - TUPLES

mytuple = ("apple", "banana", "cherry")

- Tuples are used to store multiple items in a **single variable.**
- **Tuple** is a collection which is **ordered and unchangeable.**
- Allows **duplicate members.**
- In Python tuples are written with **round brackets.**

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Tuple Items

- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has

  index [1] etc.

Ordered : When we say that tuples are ordered, it means that the items have a

defined order, and that order will not change.
Unchangeable : Tuples are unchangeable, meaning that we **cannot change, add or remove**

items after the tuple has been created.

Allow Duplicates : Since tuples are indexed, they can have items with the same value:

Example : Tuples allow duplicate values:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

Tuple Length

To determine how many items a tuple has, use the len() method:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

Create Tuple With One Item

- To create a tuple with only one item, you have **to add a comma after the item,** otherwise Python will not recognize it as a tuple.

Example : One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

Tuple Items - Data Types

- Tuple items **can be** of any data type:

Example : String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)

print(tuple1)
print(tuple2)
print(tuple3)
```

```
('apple', 'banana', 'cherry')
(1, 5, 7, 9, 3)
(True, False, False)
```

- A tuple **can contain** different data types:

Example : A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")

print(tuple1)
```

```
('abc', 34, True, 40, 'male')
```

type()

- From Python's perspective, tuples are defined as objects with the data type 'tuple':

<class 'tuple'>

Example : What is the data type of a tuple?

```
mytuple = ("apple", "banana", "cherry")

print(type(mytuple))
```

```
<class 'tuple'>
```

Python - Access Tuple Items

Access Tuple Items :

- You can access tuple items by referring to the index number, inside square brackets:

Example: Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

Note: The first item has index 0.

Negative Indexing :

- Negative indexing means beginning from **the end**
- Remember that the last item has the index -1
  - -1 refers to the **last item**
  - -2 refers to the second last item etc.

Example : Print the last item of the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Range of Indexes:

- You can specify a range of indexes by specifying where **to start and where to end the range.**
- When specifying a range, the return value will be a new tuple with the specified items.

Example : Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

•By leaving out the start value, the range will start at the first item: **Example : This example returns the items from the beginning to, but NOT included, "kiwi":**

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

```
('apple', 'banana', 'cherry', 'orange')
```

- By leaving out the end value, the range will go on to the end of the list:

Example : This example returns the items from "cherry" and to the end:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

```
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Range of Negative Indexes

- Negative indexing means starting from the end of the tuple.
- Remember that the last item has the index -1

Example : This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

```
('orange', 'kiwi', 'melon')
```

Check if Item Exists :

- To determine if a specified item is present in a tuple use the in keyword:
- **Syntax :**

if “item" in tuple:
condition

Example: Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

Python - Update Tuples

- Tuples are unchangeable, meaning that you cannot change, add, or remove items **once the tuple is created.**

Add Items

- Once a tuple is created, you **cannot add items** to it. Tuples are **unchangeable**.

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

```
Traceback (most recent call last):
  File "C:\Users\UDDISH\Downloads\Python Programs\a.py", line 2, in <module>
    thistuple[3] = "orange" # This will raise an error
TypeError: 'tuple' object does not support item assignment
>>>
```

Remove Items

Note: You cannot remove items in a tuple.

- Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

```
Traceback (most recent call last):
  File "C:\Users\UDDISH\Downloads\Python Programs\a.py", line 3, in <module>
    print(thistuple) #this will raise an error because the tuple no longer exist
s
NameError: name 'thistuple' is not defined
>>>
```

Join Two Tuples

- To join two or more tuples you can use the **+ operator:**

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Tuple Methods

- Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Python Collections - SETS

Python - SETS

myset = {"apple", "banana", "cherry"}

- Sets are used to store multiple items in a **single variable.**
- Set is a collection which is **unordered and unindexed.**
- **No duplicate** members.
- In Python sets are written with **curly brackets.**

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

```
{'apple', 'banana', 'cherry'}
```

```
{'banana', 'cherry', 'apple'}
```

Note: **Sets are unordered**

- **meaning: the items will appear in a random order.**
- **so you cannot be sure in which order the items will appear.**

Set Items

- Set items are **unordered, unchangeable, and do not allow duplicate values.**

Unordered :

- Unordered means that the items in a set do not have a defined order.
- Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable :

- Sets are unchangeable, meaning that we cannot change the items after the set has been created.

NOTE : Once a set is created, you cannot change its items, but you can add new items.

Duplicates Not Allowed:

Sets cannot have two items with the same value.

Get the Length of a Set

- To determine how many items a set has, use the len() method.

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

```
3
```

Set Items - Data Types

- Set items can be of any data type:

Example : String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}

print(set1)
print(set2)
print(set3)
```

```
{'cherry', 'apple', 'banana'}
{1, 3, 5, 7, 9}
{False, True}
```

- A set can contain different data types:

Example : A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}

print(set1)
```

```
{True, 34, 40, 'male', 'abc'}
```

type()

- From Python's perspective, sets are defined as objects with the data type 'set':

  <class 'set'>

Example : What is the data type of a set?

```
myset = {"apple", "banana", "cherry"}

print(type(myset))
```

```
<class 'set'>
```

Python - Access Items

- You **cannot access items** in a set by referring to an index, since sets are unordered the items has no index.
- But you can **loop through** the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example:
Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
apple
cherry
banana
```

Example:
Check if "banana" is present in the set:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

Change Items

- Once a set is created, **you cannot change** its items, but you **can add** new items.

Python - Add Set Items

Add Items:

- Once a set is created, you cannot change its items, **but you can add new items.**
- To add one item to a set use the add() method.
- To add more than one item to a set use the update() method.

Example: Add an item to a set, using the add() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'banana', 'orange', 'cherry', 'apple'}
```

Example: Add multiple items to a set, using the update() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

```
{'orange', 'mango', 'cherry', 'grapes', 'banana', 'apple'}
```

Python - Remove Item

- To remove an item in a set, use the remove(), or the discard() method.

Example:

Remove "banana" by using the remove() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

Example:

Remove "banana" by using the discard() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

Note: If the item to remove does not exist,

- **remove() will raise an error.**
- **discard() will NOT raise an error.**

Join Two Sets

- There are several ways to join two or more sets in Python.
- You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

Example : The union()/update() method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{2, 3, 1, 'c', 'a', 'b'}
```

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{'c', 'a', 'b', 2, 1, 3}
```

Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Python Collections - Dictionaries

Python - Dictionaries

- Dictionary is a collection which is **ordered, indexed and changeable.**
- **No duplicate** members.
- In Python Dictionary are written with **curly brackets** and have **keys and values:**

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Dictionary Items

- Dictionary items are ordered, changeable, and do not allow duplicate values.
- Dictionary items are indexed, the first item has index [0], the second item has index [1] etc.

- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
Ford
```

- **Ordered :** When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
- **Changeable :** Dictionaries are changeable, meaning that we **can change, add or remove** items after the dictionary has been created.
- **Duplicates Not Allowed :** Dictionaries cannot have two items with the same key:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Dictionary Length

- To determine how many items a dictionary has, use the len() function:

Example : Print the number of items in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 2020
}
print(len(thisdict))
```

```
3
```

Dictionary Items - Data Types

- Dictionary items can be of any data type:

Example : String, int, boolean, and list data types:

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

print(thisdict)
```

```
{'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red', 'white', 'blue']}
```

type()

- From Python's perspective, dictionaries are defined as objects with the data type 'dict':

  <class 'dict'>

Example : Print the data type of a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

```
<class 'dict'>
```

Python - Access Dictionary Items

Get Items :

- The items() method will return each item in a dictionary, as tuples in a list.

Example : Get a list of the key:value pairs

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.items()

print(x)
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

Get Keys :

- The keys() method will return a list of all the keys in the dictionary.

Example : Get a list of the keys:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.keys()

print(x)
```

```
dict_keys(['brand', 'model', 'year'])
```

Get Values:

- The values() method will return a list of all the values in the dictionary.

Example : Get a list of the values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.values()

print(x)
```

```
dict_values(['Ford', 'Mustang', 1964])
```

Accessing Items

- You can access the items of a dictionary by referring to **its key name, inside square brackets:**

Example : Get the value of the "model" key:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

```
Mustang
```

- **There is also a method called get() that will give you the same result:**

Example : Get the value of the "model" key:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

```
Mustang
```

Python - Change Dictionary Items

Change Values :

- You can change the value of a specific item by referring to its **key name:**

Example : Change the "year" to 2018:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict["year"] = 2018

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Update Dictionary :

- The update() method will update the dictionary with the items from the given argument.
- The argument must be a dictionary, or an iterable object with **key:value pairs.**

Example : Update the "year" of the car by using the update() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Python - Add Dictionary Items

Adding Items :

- Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example :

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Update Dictionary :

- The update() method will update the dictionary with the items from a given argument.

  If the item does not exist, the item will be added.

- The argument must be a dictionary, or an iterable object with key:value pairs.

Example : Add a color item to the dictionary by using the update() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Python - Remove Dictionary Items

Removing Items :

- There are several methods to remove items from a dictionary:

pop() method :

- The pop() method removes the item with **the specified key name**:

Example :

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

popitem() method

- The popitem() method removes **the last inserted item**

Example :

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

del() method

- The del keyword removes the item with the **specified key name:**

Example :

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

- The del keyword can also delete the **dictionary completely**:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

```
Traceback (most recent call last):
  File "demo_dictionary_del3.py", line 7, in <module>
    print(thisdict) #this will cause an error because "thisdict" no longer exists.
NameError: name 'thisdict' is not defined
```

clear() method

- The clear() method empties the dictionary:

Example:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
{}
```

Python File - Handling

Python File - Handling

- File handling is an important part of any web application.
- Python has several functions for creating, reading, updating, and deleting files.

Python - File Handling

The key function for working with files in Python is the **open() function.**

- The open() function takes **two parameters; *filename*, and *mode*.**

There are **four different methods** (modes) for opening a file:

- **"r" - Read** - Default value. Opens a file for reading, error if the file does not exist
- **"a" - Append** - Opens a file for appending, creates the file if it does not exist
- **"w" - Write** - Opens a file for writing, creates the file if it does not exist
- **"x" - Create** - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

- **"t" - Text** - Default value. Text mode
- **"b" - Binary** - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

- Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Python File Open - Open a File on the Server

- **Assume we have the following file, located in the same folder as Python:**

- To open the file, use the built-in open() function.
- The open() function returns a file object, which has a read() method for reading the content of the file:

demofile.txt

```
WELCOME
TO  PYTHON
CLASS
```

Ex – 1:

```
f = open("demofile.txt", "r")
print(f.read())
```

```
======== RESTART: E:\K4U TRANNING'
WELCOME
TO
PYTHON
CLASS
```

If the **file is located in a different location**, you will have to specify the file path, like this:

EX - 2 : Open a file on a different location:

```
f = open("E:\K4U\K4U TRANNING\Python Programs\welcome.txt", "r")

print(f.read())
```

```
======= RESTART: E:\K4U TRANNING\Python Programs\File Handling\2 file.py =
WELCOME to Python CLASS
>>>
```

Read Only Parts of the File

By default the read() method returns the **whole text,** but you can also specify how **many characters you want to return:**

Ex – 3 : Return the 5 first characters of the file

```
f = open("demofile.txt", "r")
print(f.read(5))
```

```
=== RESTART: E:\K4
WELCO
>>>
```

demofile.txt

```
WELCOME
TO  PYTHON
CLASS
```

Read Lines - readline() method:

You can return **one line** by using the **readline()** method:

Ex – 4 : Read one line of the file f =

```
open("demofile.txt", "r")
print(f.readline())
```

```
======== RESTART: E:\
WELCOME
```

By calling readline() two times, you can read the two first lines:

Ex – 5 : Read two lines of the file: f =

```
open("demofile.txt", "r")  print(f.readline())
print(f.readline())
```

```
======== RESTART: E:\K4U TRANN
WELCOME

TO
```

read the whole file, line by line

By looping through the lines of the file, you can **read the whole file, line by line:**

Ex -6 : Loop through the file line by line

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Close Files

It is a good practice to always close the file when you are done with it.

Ex : Close the file when you are finish with it

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

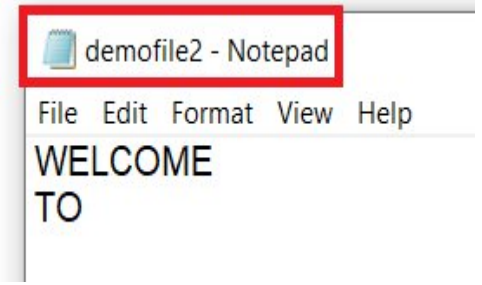
Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Python File Write - Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - **Append** - will append to the end of the file

"w" - **Write** - will overwrite any existing content



Ex – 7 : Open the file "demofile2.txt" and append content to the file

```
f = open("demofile2.txt", "a")
f.write("B Prasad Python class!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

```
======= RESTART: E:\K4U T
WELCOME
TO
B Prasad Python class!
>>>
```

Python File Write - overwrite the content

Ex – 8 : Open the file "demofile3.txt" and overwrite the content

```
f = open("demofile3.txt", "w")
f.write(" Prasad - Add content!")
f.close()

#open and read the file after the appending: f =
open("demofile3.txt", "r")
print(f.read())
```

```
====== RESTART: E:\K4U TR
Prasad - Add content!
```

Note: the **"w" method** will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

- **"x" - Create** - will create a file, returns an error if the file exist
- **"a" - Append** - will create a file if the specified file does not exist
- **"w" - Write** - will create a file if the specified file does not exist

Example:

Create a file called "myfile.txt": f =

```
open("myfile.txt", "x")
```

Result: a new empty file is created!

Example:

Create a new file if it does not exist: f =

```
open("myfile.txt", "w")
```

Python Delete File - Delete a File

To delete a file, you must import the **OS module,** and run its **os.remove() function:**

Ex - 10: Remove the file "demofile4.txt"

```
import os

os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Ex – 11 : Check if file exists, *then* delete it

```
import os
if os.path.exists("demofile4.txt"):
  os.remove("demofile4.txt")

else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the os.rmdir() method:

Ex - Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

Python - User Input

User Input

- **Python allows for user input.**
- **That means we are able to ask the user for input.**

Python input() Function :

- The input() function allows user input.

Syntax : input(*prompt*)

Example 1:

```
name = input("Enter your name: ")
print(name)
```

```
Enter your name: PRASAD
PRASAD
```

Example 2:

```
username = input("Enter username:")
print("Username is: " + username)
```

```
Enter username:prasad
Username is: prasad
```

Example 3:

```
x = input("Enter First Number: ")
y = input("Enter Second Number: ")

sum = x + y

print("The sum is: ", sum)
```

```
========== RESTART: C:\Users\U
Enter First Number: 5
Enter Second Number: 6
The sum is:  56
```

Example 4:

```
x = input("Enter First String: ")
y = input("Enter Second String: ")

Concatinate = x + y

print("The Total String is: ", Concatinate)
```

```
================== RESTART: C:\Users\UDDISH\Do
Enter First String: Welcome To
Enter Second String:  Python Class
The Total String is:  Welcome To Python Class
>>>
```

Python Strings

Python - String

- String literals in python are surrounded by either single quotation marks, or double quotation marks.

ex: 'hello' is the same as "hello".

You can display a string literal with the print() function:

```
print("Hello")
print('Hello')
```

```
Hello
Hello
```

Assign String to a Variable

- Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

```
a = "Hello"
print(a)
```

```
Hello
```

Multiline Strings

- You can assign a multiline string to a variable by using **three quotes (single or double):**

Three double quotes: “ “ “ “ “ “

Example : You can use three double quotes:

```
a = """start with a letter
cannot start with a number
contain alpha numeric characters
case-sensitive """
print(a)
```

```
start with a letter
cannot start with a number
contain alpha numeric characters
case-sensitive
```

Three single quotes: ‘’’ ‘’’

Example : You can use three single quotes:

```
a = '''start with a letter
cannot start with a number
contain alpha numeric characters
case-sensitive '''
print(a)
```

```
start with a letter
cannot start with a number
contain alpha numeric characters
case-sensitive
```

Python - Slicing Strings

Python - Modify Strings

- Python has a set of **built-in methods** that you can use on strings.

Upper Case : **upper() method**

Example : The upper() method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

```
HELLO, WORLD!
```

Lower Case : **lower() method**

Example : The lower() method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

```
hello, world!
```

Remove Whitespace

- Whitespace is the space before and/or after the actual text, and very often you want to remove this space.
- The **strip() method** removes any whitespace from the beginning or the end:

Example :

```
a = " Hello, World! "
print(a.strip())
```

```
Hello, World!
```

Replace String

- The replace() method replaces a string with another string:

Example :

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
Jello, World!
```

Split String

- The split() method returns a list where the text between the specified separator becomes the list items.
- The **split()** method **splits the string into substrings** if it finds **instances of the separator:**

```
a = "Hello, World!"
b = a.split(",")
print(b)
```

```
['Hello', ' World!']
```

```
a = "Hello + World"
b = a.split("+")
print(b)
```

```
['Hello ', ' World']
```

```
a = "Welcome to Python Class"
b = a.split("to")
print(b)
```

```
['Welcome ', ' Python Class']
```

```
a = "Welcome to Python to Class"
b = a.split("to")
print(b)
```

```
['Welcome ', ' Python ', ' Class']
```

- The capitalize() method returns a string where the first character is upper case, and the rest is lower case.

```
txt = "hello, and welcome to my world."

x = txt.capitalize()

print (x)
```

```
Hello, and welcome to my world.
```

- The casefold() method returns a string where all the characters are lower case.

```
txt = "Hello, And Welcome To My World!"

x = txt.casefold()

print(x)
```

```
hello, and welcome to my world!
```

Python - Format - Strings

- Python Variables **cannot combine** strings and numbers :

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

```
Traceback (most recent call last):
  File "demo_string_format_error.py", line 2, in <module>
    txt = "My name is John, I am " + age
TypeError: must be str, not int
```

String Format

- But we can **combine strings and numbers** by using the format() method!
- The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Example : Use the format() method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

```
My name is John, and I am 36
```

- The format() method takes **unlimited number** of arguments, and are placed into the respective placeholders:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

- You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567
```

Python - Escape Characters

Escape Character :

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

```
txt = "Welocme to "PYTHON" class."
```

- To fix this problem, use the escape character \":

```
txt = "Welocme to \"PYTHON\" class."
print(txt)
```

```
Welocme to "PYTHON" class.
```


Thank You