Introduction to Python

- NIELIT CHANDIGARH

Python – a mysterious name

Python is a widely used general-purpose, high level programming language. It was initially designed by Dutch programmer **Guido van Rossum in 1991.**

The name Python comes from an old BBC television comedy sketch series called Monty Python’s Flying Circus. When Guido van Rossum was creating Python, he was also reading the scripts of Monty Python. He thought the name Python was appropriately short and slightly mysterious.

Python was conceived in the late 1980s, & implementation began in December 1989 by **Guido van Rossum**

Why should we learn Python?

- Python is a high-level programming language.
- Its syntax allows programmers to express concepts in fewer lines of code.
- Python is a programming language that lets you work quickly and integrate systems more efficiently.
- One can plot figures using Python.
- One can perform symbolic mathematics easily using Python.
- It is available freely online.

Python versions

Python was first released on February 20, 1991, and later developed by Python Software Foundation.

Major Python versions are – **Python 1**, **Python 2 and Python 3**.

- On 26th January 1994, **Python 1.0** was released.
- On 16th October 2000, **Python 2.0** was released with many new features.
- On 3rd December 2008, **Python 3.0** was released with more testing and includes new features.

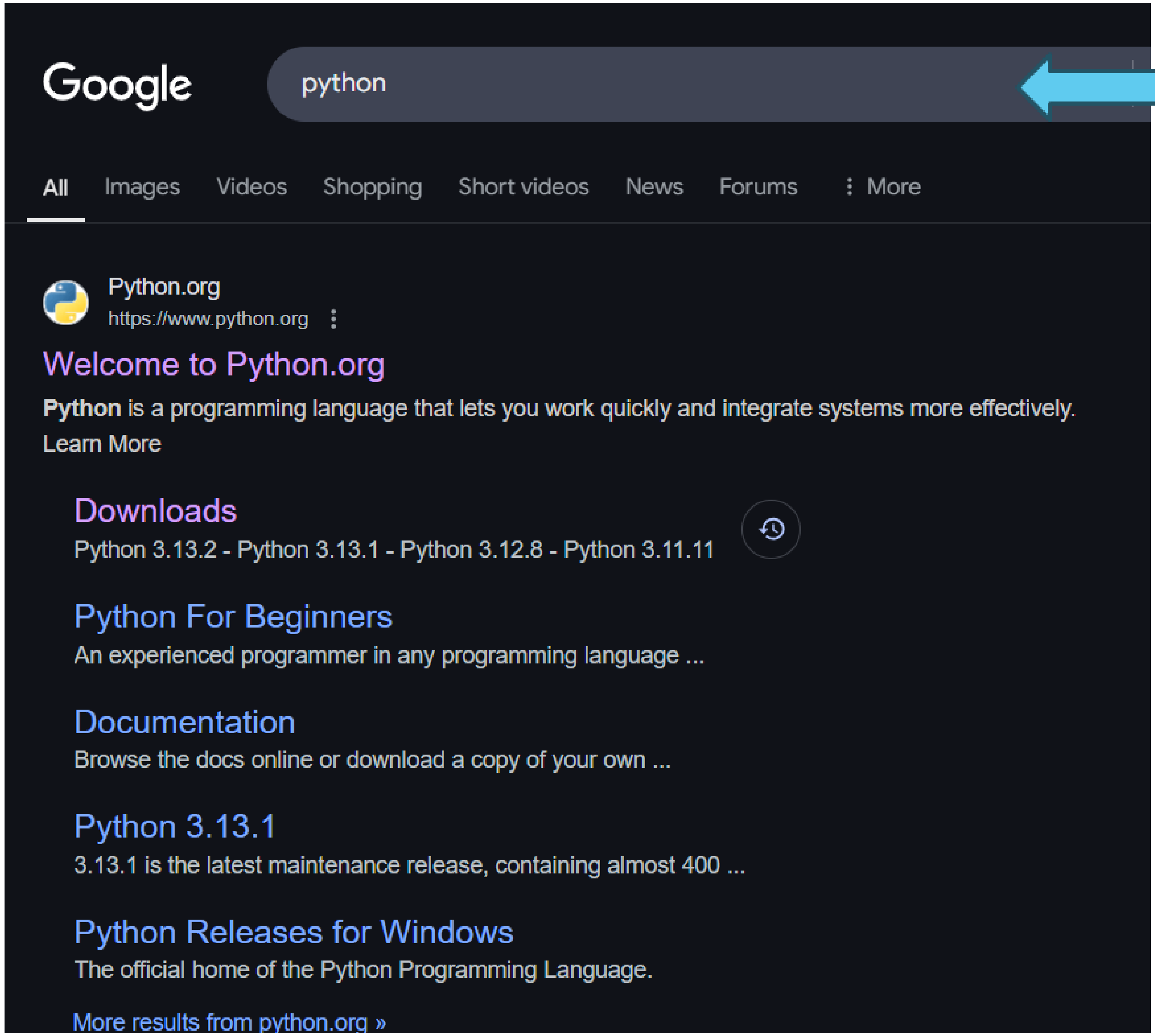
Latest version - On 4TH February 2025, **Python 3.13.2** was released.

To check your Python version :

i) `python --version` in the terminal window.

ii) To check your Python version using the sys module. Run the following command inside a Python interpreter or script:

```
import sys
print(sys.version)
```


Google

python

All Images Videos Shopping Short videos News Forums ⋮ More

Python.org
https://www.python.org

Welcome to Python.org

Python is a programming language that lets you work quickly and integrate systems more effectively.
Learn More

Downloads
Python 3.13.2 - Python 3.13.1 - Python 3.12.8 - Python 3.11.11

Python For Beginners
An experienced programmer in any programming language ...

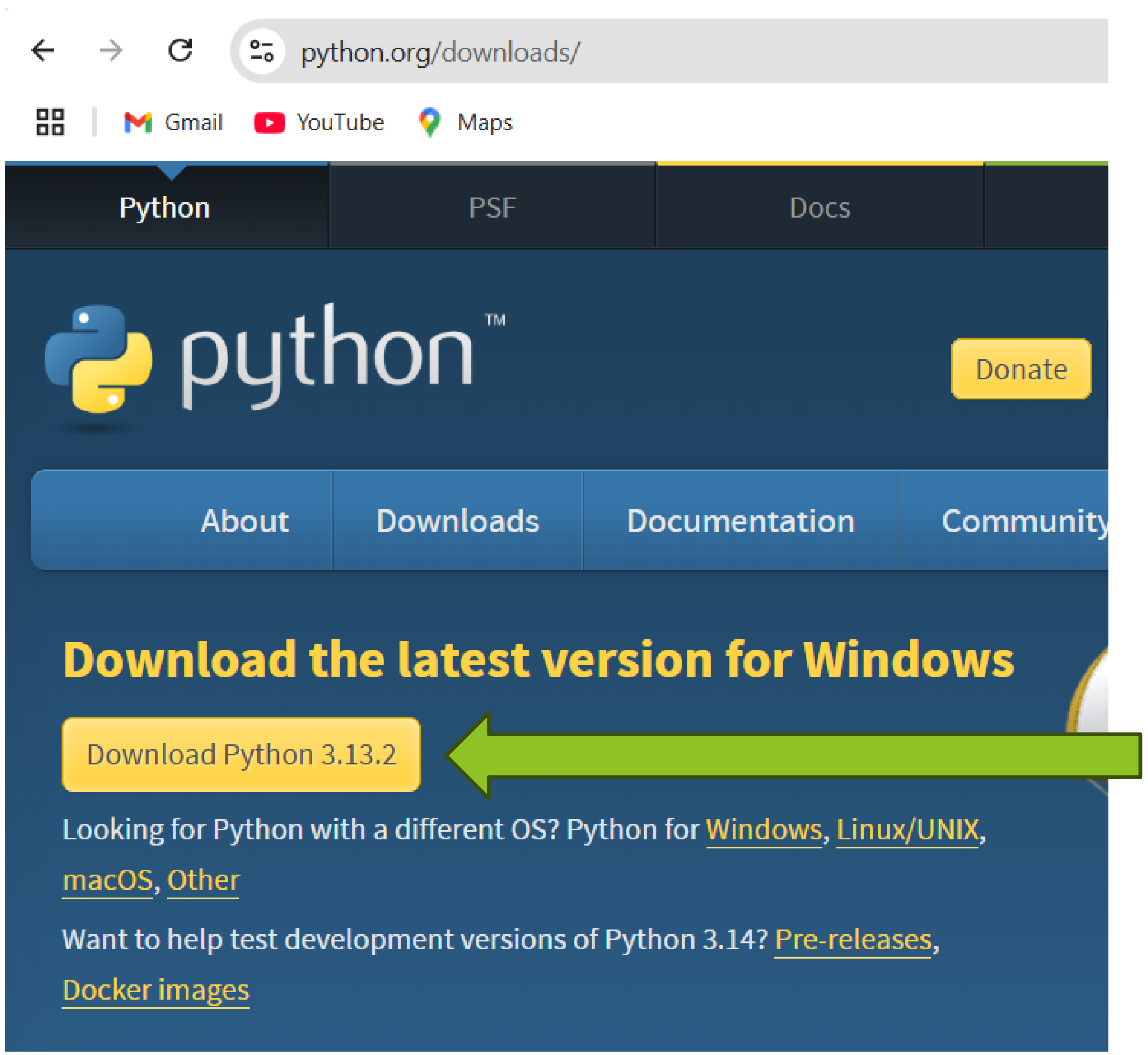
Documentation
Browse the docs online or download a copy of your own ...

Python 3.13.1
3.13.1 is the latest maintenance release, containing almost 400 ...

Python Releases for Windows
The official home of the Python Programming Language.

More results from python.org »

Searching for Python



Download the latest version for Windows

Download Python 3.13.2

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.14? Pre-releases, Docker images

Downloading Python

Python Interpreter

The program that translates Python instructions and then executes them is the Python interpreter. When we write a Python program, the program is executed by the Python interpreter. This interpreter is written in the C language.

There are certain online interpreters like

https://ide.geeksforgeeks.org/,

http://ideone.com/ ,

http://codepad.org/

W3Schools Python Compiler: https://www.w3schools.com/python/python_compiler.asp

Google Colab – A cloud-based Jupyter Notebook environment that allows users to write and execute Python code in a web browser, with free access to GPUs and TPUs.

that can be used to start Python without installing an interpreter.

Python IDLE

Python interpreter is embedded in several larger programs that make it particularly easy to develop Python programs. Such a programming environment is IDLE

(**Integrated Development and Learning Environment**).

It is available freely online. For Windows machine IDLE (Integrated Development and Learning Environment) is installed when you install Python.

Running Python

There are two modes for using the Python interpreter:

1) Interactive Mode - REPL (Read-Eval-Print Loop)

2) Script Mode - This is when you write Python code in a .py file and run it as a script instead of entering commands interactively.

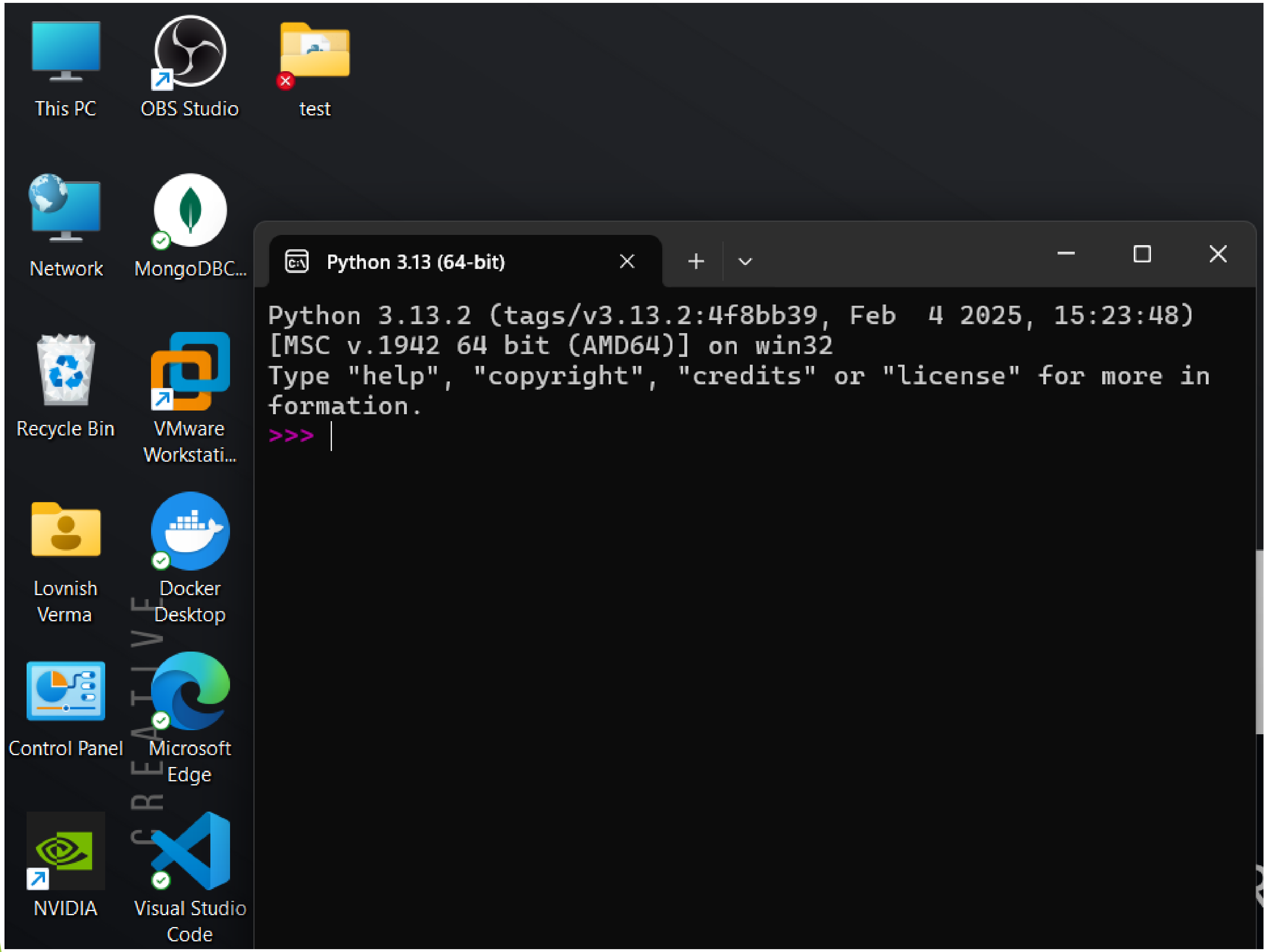
Options for running the program:

- Windows: Double-click madlib.py or run in Command Prompt

```
python madlib.py
```

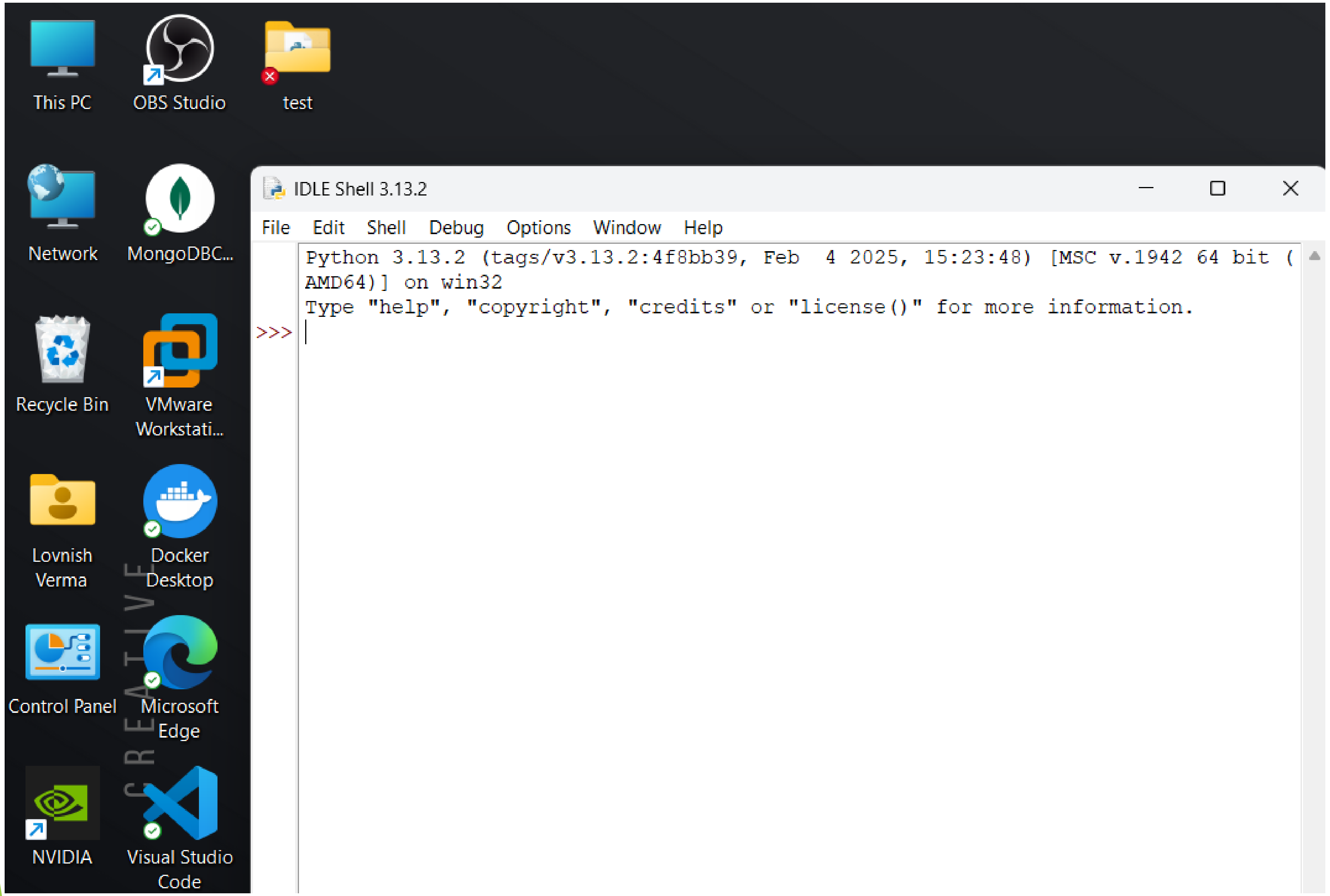
- Linux/macOS: Open a terminal, navigate to the directory, and type:

```
python madlib.py
```



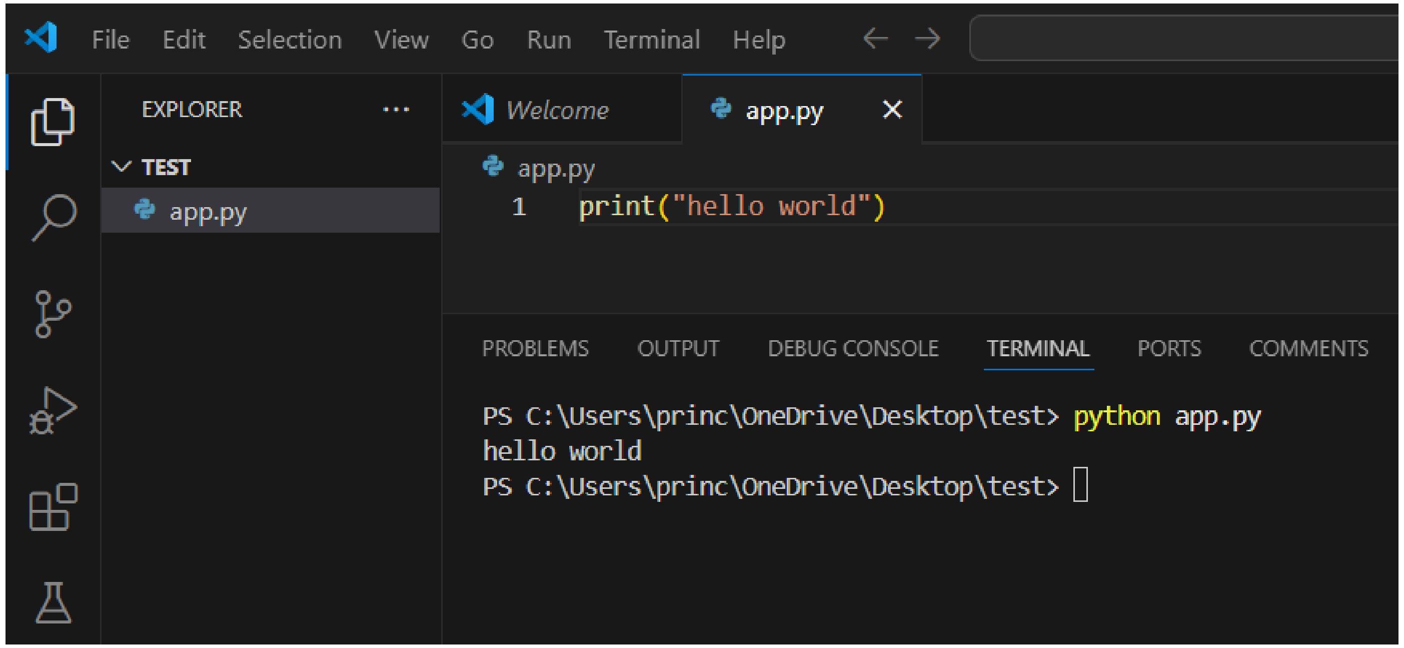
```
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48)
[MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> 
```

Interactive shell



```
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

IDLE shell



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

```
PS C:\Users\princ\OneDrive\Desktop\test> python app.py
hello world
PS C:\Users\princ\OneDrive\Desktop\test>
```

Visual Studio Code (code editor)

Running Python

1) in interactive mode:

```
>>> print("Hello Teachers")
 Hello Teachers
 >>> a=10
>>> print(a)
10
>>> x=10
>>> z=x+20
>>> z
30
```

Running Python

2) in script mode:

Programmers can store Python script source code in a file with the .py extension, and use the interpreter to execute the contents of the file.

For Windows and LINUX OS to run a script file YourFile.py you have to type:

```
python YourFile.py
```

Data Types

Python has various standard data types:

▪ Integer	[class ‘int’]
▪ Float	[class ‘float’]
▪ Boolean	[class ‘bool’]
▪ String	[class ‘str’]

Integer

Int:

For integer or whole number, positive or negative, without decimals of unlimited length.

```
>>> print(2465635468765)
2465635468765
 >>> print(0b10)          # 0b indicates binary number
2
>>> print(0x10)           # 0x indicates hexadecimal number
16
 >>> a=11
>>> print(type(a))
<class 'int'>
```

Float

Float:

Float, or "floating point number" is a number, positive or negative.

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
 >>> y=2.8
>>> y
2.8
>>> print(0.00000045)
4.5e-07 #e-07 → This represents "× 10⁻⁷" (which means move the decimal point 7 places to the left).
>>> y=2.8
>>> print(type(y))
<class 'float'>
```

Boolean and String

Boolean:

Objects of Boolean type may have one of two values, True or False:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

String:

```
>>> print(‘Science college’)
Science college
>>> type("My college")
<class 'str'>
```

Variables

One can store integers, decimals or characters in variables.

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)

```
a= 100              # An integer assignment
b = 1040.23         # A floating point
c = "John"          # A string
```

List, Tuple, Set, Dictionary

Four built-in data structures in Python:-
list, tuple, set, dictionary
- each having qualities and usage different from the other three.

List is a collection of items that is written with square brackets. It is mutable, ordered and allows duplicate members. Example: **list = [1,2,3,'A','B',7,8,[10,11]]**

Tuple is a collection of objects that is written with first brackets. It is immutable. Example: **tuple = (2, 1, 10, 4, 7)**

Set is a collection of elements that is written with curly brackets. It is unindexed and unordered. Example: **S = {x for x in 'abracadabra' if x not in 'abc'}**

Dictionary is a collection which is ordered, changeable and does not allow duplicates. It is written with curly brackets and objects are stored in key: value format.
Example: **X = {1:'A', 2:'B', 3:'c'}**

print function

```
>>>type(print)
```

Output:

```
builtin_function_or_method
```

```
>>>print( ‘Good morning’ )  or  print(“Good morning”)
```

Output:

```
Good morning
```

```
>>>print(“Workshop”, “on”, “Python”)  or  print(“Workshop on Python”)
```

Output:

```
Workshop on Python
```

print function

```
>>>print(‘Workshop’, ‘on’, ‘Python’, sep=’\n’)
```

```
# sep=‘\n’ will put each word in a new line
```

Output:

```
Workshop
 on
Python
```

```
>>>print(‘Workshop’, ‘on’, ‘Python’, sep=’, ’)
```

```
# sep=‘, ’ will print words separated by ,
```

Output:

```
Workshop, on, Python
```

print function

%d is used as a placeholder for integer value.

%f is used as a placeholder for decimal value.

%s is used as a placeholder for string.

```
a = 2

b = ‘tiger’

print(a, ‘is an integer while’, b, ‘is a string.’)
```

Output:

```
        2 is an integer while tiger  is a string.
```

Alternative way:

```
print(“%d is an integer while %s is a string.”%(a, b))
```

Output:

```
         2 is an integer while tiger  is a string.
```

print function

# printing a string

name = “Rahul”

print(“Hey ” + name)

Output:

Hey Rahul

print("Roll No: " + str(45)) # Output: Roll No: 45

print("Roll No: 45") # Output: Roll No: 45

print("Roll No:", 45) # Output: Roll No: 45

Output:

Roll No: 34

# printing a bool True / False

print(True)

Output: True

# Using f-Strings (Modern & Recommended)

#Example using f-strings:

name = "Rahul"

print(f"Hey {name}")

Output: Hey Rahul

# Note: Python 3.6+ introduced f-strings, which make string formatting much easier and more readable.

#Not ❌

print("Roll No: " + 45)

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

print function

int_list = [1, 2, 3, 4, 5]

print(int_list)   **# printing a list**

Output: [1, 2, 3, 4, 5]

my_tuple = (10, 20, 30)

print(my_tuple)   **# printing a tuple**

Output: (10, 20, 30)

my_dict = {“language”: “Python”, “field”: “data science”}

print(my_dict)   **# printing a dictionary**

Output: {“language”: “Python”, “field”: “data science”}

my_set = {“red”, “yellow”, “green”, “blue”}

print(my_set)   **#printing a set**

Output: {“red”, “yellow”, “green”, “blue”}

print function

```
str1 = ‘Python code’
str2 = ‘Matlab code’
print(str1)
print(str2)
```

Output:

```
Python code
Matlab code
```

```
print(str1, end=’ ‘)
print(str2)
```

Output:

```
Python code Matlab code
```

```
print(str1, end=’, ‘)
print(str2)
```

Output:

```
Python code, Matlab code
```

print function

```
items = [ 1, 2, 3, 4, 5]
for item in items:
    print(item)
```

Output:

```
1
2
3
4
5
```

```
items = [ 1, 2, 3, 4, 5]
for item in items:
    print(item, end=’  ‘)
```

Output:

```
1  2  3  4  5
```

Operators

Addition	+	Subtraction	-
Multiplication	*	Exponentiation	**
Division	/	Integer division	/ /
Remainder	%		
Binary left shift	<<	Binary right shift	>>
And	&	Or	|
Less than	<	Greater than	>
Less than or equal to	<=	Greater than or equal to	>=
Check equality	==	Check not equal	!=

Precedence of operators

Parenthesized expression	(…..)
Exponentiation	**
Positive, negative, bitwise not	+n, -n, ~n
Multiplication, float division, int division, remainder	*, /, //, %
Addition, subtraction	+, -
Bitwise left, right shifts	<<, >>
Bitwise and	&
Bitwise or	|
Membership and equality tests	in, not in, is, is
not, <, <=, >, >=, !=, ==	
Boolean (logical) not	not x
Boolean and	and
Boolean or	or
Conditional expression	if ….. else

Precedence of Operators

Examples:

```
a = 20
b = 10
c = 15
d = 5
e = 2
f = (a + b) * c / d
print( f)
g = a + (b * c) / d - e
print(g)
h = a + b*c**e
print(h)
```

Multiple Assignment

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

```
a = b = c = 1.5
```

```
a, b, c = 1, 2, " Red“
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively and one string object with the value "Red" is assigned to the variable c.

Special Use of + and *

Examples:

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

Output:

```
Python is awesome.
```

```
print(‘It is’ + 2*’very ’ + ’hot.’)
```

Output:

```
It is very very hot.
```

Use of \”, \n, \t

Specifying a backslash (\) in front of the quote character in a string “escapes” it and causes Python to suppress its usual special meaning. It is then interpreted simply as a literal single quote character:

```
>>> print(" \”Beauty of  Flower\” ")
”Beauty of  Flower”
>>> print('Red  \n Blue \n Green  ')
Red
Blue
Green
>>> print("a \t  b \t  c \t  d")
a       b       c       d
```

Comments

Single-line comments begins with a hash (#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of line.

A Multi line comment is useful when we need to comment on many lines. In python, triple double quote(“ “ “) and single quote(‘ ‘ ‘)are used for multi-line commenting.
Example:

```
 “““““ My Program to find

      Average of three numbers ”””””

a  =  29        #  Assigning value of a

b  =  17       #  Assigning value of b

c  =  36       #  Assigning value of c

average = ( a + b + c)/3

print(“Average value is ”, average)
```

id( ) function, ord( ) function

id( ) function: It is a built-in function that returns the unique identifier of an object. The identifier is an integer, which represents the memory address of the object. The id( ) function is commonly used to check if two variables or objects refer to the same memory location.

```
>>> a=5
>>> id(a)
1403804521000552
```

```
#Checking if Two Variables Have the Same ID
x = 10
y = x  # y refers to the same object as x
print(id(x))  # Example: 1403804521000600
print(id(y))  # Same as x: 1403804521000600
```

ord( ) function: It is used to convert a single unicode character into its integer representation.

```
>>> ord('A')   #Every character has a unique Unicode (ASCII) value.- A has 65
65
>>> chr(65) #The chr() function converts an integer (Unicode code) back to a character.
'A'
```

Control Flow Structures

1. Conditional if (if)
2. Alternative if (if else)
3. Chained Conditional if (if elif else)
4. While loop
5. For loop

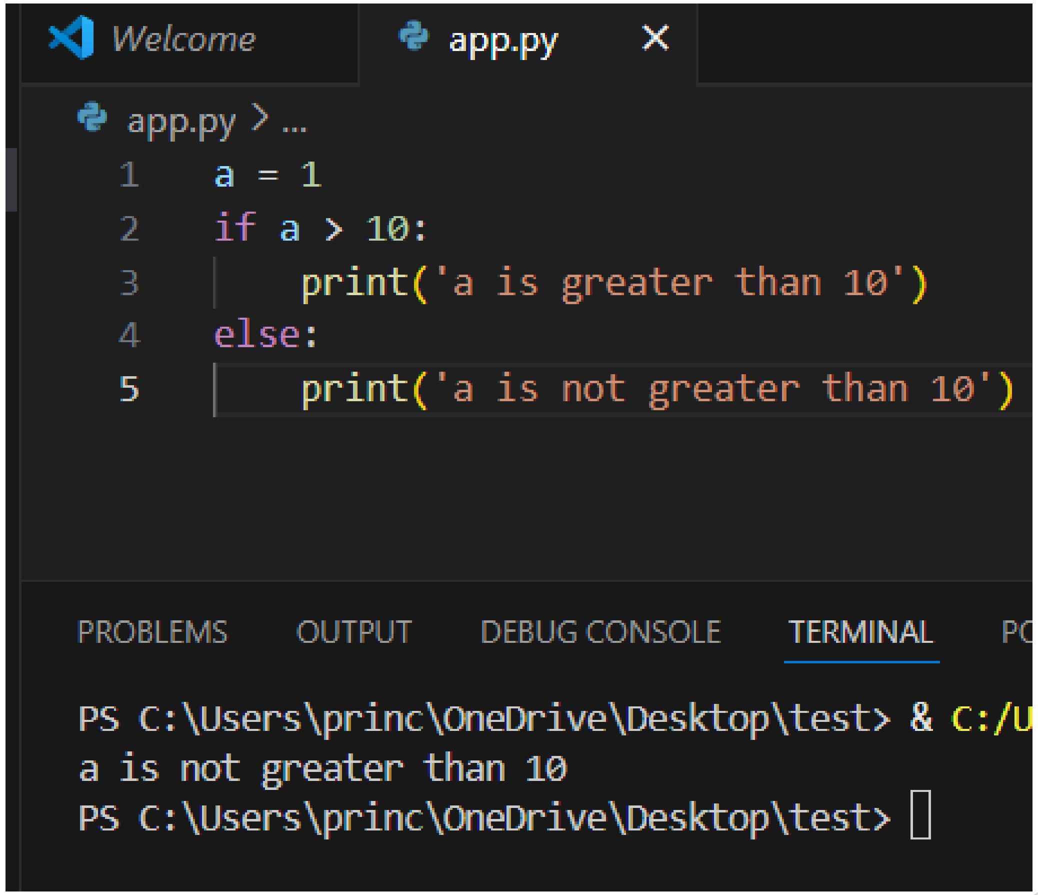
Conditional if

Example:

```
a=10

if a > 9 :

    print("a is greater than 9")
```

Output:

```
a is greater than 9
```



```
a = 1
if a > 10:
    print('a is greater than 10')
else:
    print('a is not greater than 10')
```

```
PS C:\Users\princ\OneDrive\Desktop\test> & C:/U
a is not greater than 10
PS C:\Users\princ\OneDrive\Desktop\test>
```

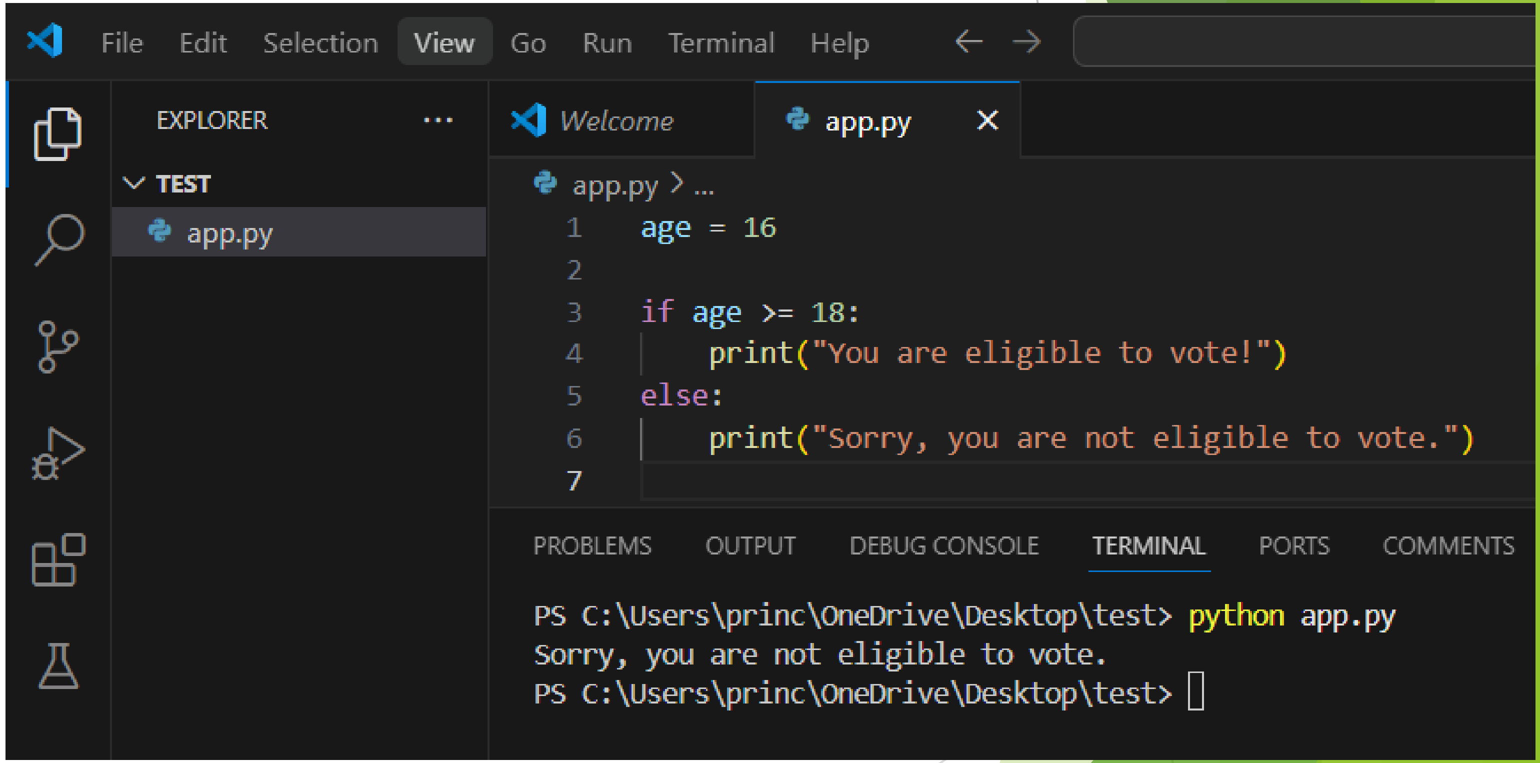
Alternative if

Example:

```
A = int(input('Enter the marks '))
if  A >= 45:
     print("PASS")
else:
     print("FAIL")
```

Output:

```
Enter the marks 75

PASS
```



```
age = 16

if age >= 18:
    print("You are eligible to vote!")
else:
    print("Sorry, you are not eligible to vote.")
```

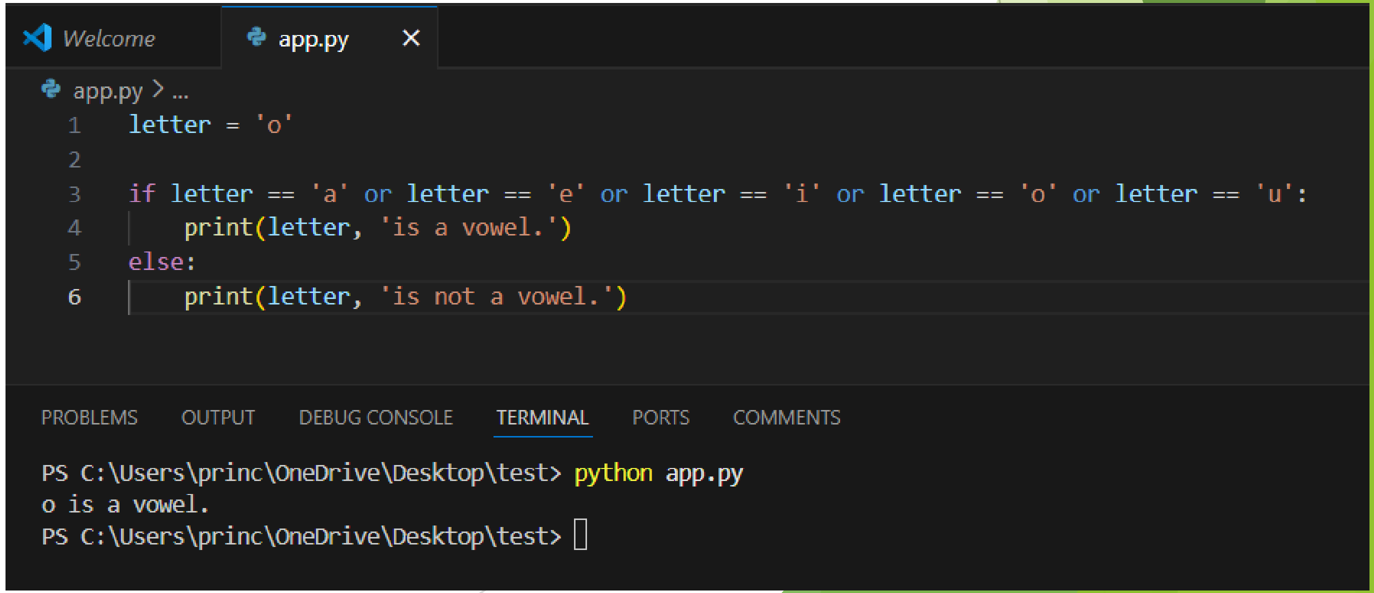
```
PS C:\Users\princ\OneDrive\Desktop\test> python app.py
Sorry, you are not eligible to vote.
PS C:\Users\princ\OneDrive\Desktop\test>
```

# Test if the given letter is vowel or not

```
letter = ‘o’
if letter  == ‘a’ or letter  == ‘e’ or letter  == ‘i’ or letter  == ‘o’
        or letter  == ‘u’ :
        print(letter, ‘is a vowel.’)
else:
        print(letter, ‘is not a vowel.’)
```

Output:

```
o is a vowel.
```



```python
letter = 'o'

if letter == 'a' or letter == 'e' or letter == 'i' or letter == 'o' or letter == 'u':
    print(letter, 'is a vowel.')
else:
    print(letter, 'is not a vowel.')
```

```
PS C:\Users\princ\OneDrive\Desktop\test> python app.py
o is a vowel.
PS C:\Users\princ\OneDrive\Desktop\test>
```

# Program to find the greatest of three different numbers

```
a = int(input('Enter 1st no’))

b = int(input('Enter 2nd no'))

c= int(input('Enter 3rd no'))

if a > b:

    if a>c:

        print('The greatest no is ', a)

    else:

        print('The greatest no is ', c)

else:

    if b>c:

        print('The greatest no is ', b)

    else:

      print('The greatest no is ', c)
```

Output:

```
Enter 1st no 12

Enter 2nd no 31

Enter 3rd no 9

The greatest no is  31
```

Chained conditional if

# Program to guess the vegetable

```
 color = “green”
 if  color == “red”:
        print(‘It is a tomato.’)
elif  color ==  “purple”:
        print(‘It is a brinjal.')
elif  color == “green”:
        print(‘It is a papaya. ')
else:
        print(‘There is no such vegetable.’)
```

Output:

```
            It is a papaya.
```

# Program to find out the greatest of four different numbers

```
a=int(input('Enter 1st no ‘))
b=int(input('Enter 2nd no ‘))
c=int(input('Enter 3rd no ‘))
d=int(input('Enter 4th no ‘))
if (a>b and a>c and a>d):
    print('The greatest no is ', a)
elif (b>c and b>d):
    print('The greatest no is ', b)
elif (c>d):
    print('The greatest no is ', c)
elif d>c :
    print('The greatest no is ', d)
else:
    print('At least two values are equal')
```

Output:

```
Enter 1st no 23
Enter 2nd no 10
Enter 3rd no 34
Enter 4th no 7
The greatest no is  34
```

# Program to find out Grade

```
marks = int(input('Enter total marks ‘))

total = 500     # Total marks

percentage=(marks/total)*100

if percentage >= 80:

    print('Grade O')

elif percentage >=70:

    print('Grade A')

elif percentage >=60:

    print('Grade B')

elif percentage >=40:

    print('Grade C')

else:

    print('Fail‘)
```

Output:

```
Enter total marks 312

Grade B
```

While loop

The while loop in Python is used when we don't know the exact number of iterations beforehand. It keeps running as long as a condition is True.

Basic Syntax:

```
while condition:    # Code to execute while condition is True
```

Example: *Printing Numbers from 1 to 5*

```
i = 1  # Initialization
while i <= 5:  # Condition
    print(i)
    i += 1  # Increment
```

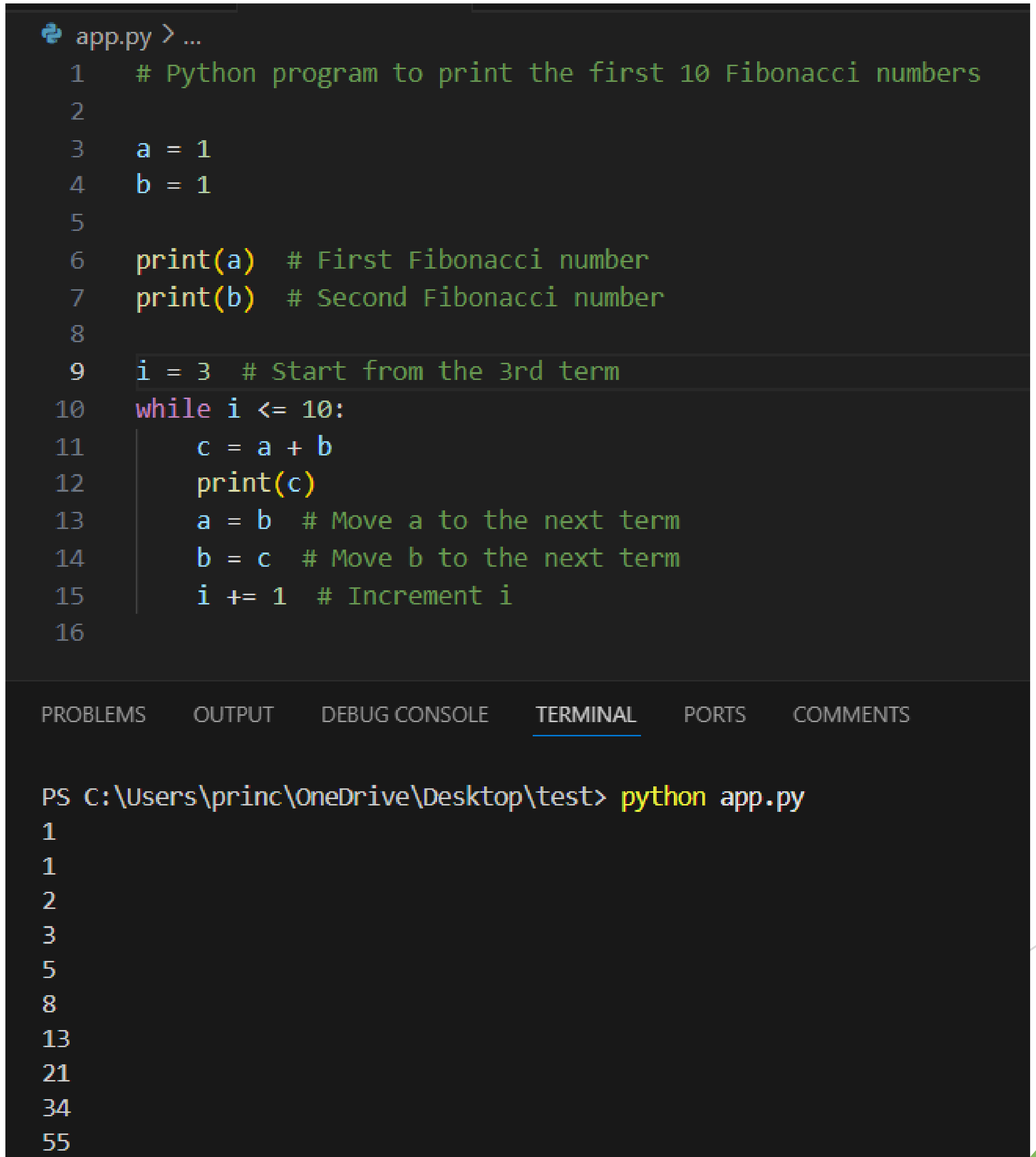
Output:

```
1
2
3
4
5
```

While loop

# Python program to find first ten Fibonacci numbers

```
a=1
print(a)
b=1
print(b)
i=3
while i<= 10:
    c=a+b
    print(c)
    a=b
    b=c
    i=i+1
```



app.py > ...

```python
# Python program to print the first 10 Fibonacci numbers

a = 1
b = 1

print(a)  # First Fibonacci number
print(b)  # Second Fibonacci number

i = 3  # Start from the 3rd term
while i <= 10:
    c = a + b
    print(c)
    a = b  # Move a to the next term
    b = c  # Move b to the next term
    i += 1  # Increment i

```

```
PS C:\Users\princ\OneDrive\Desktop\test> python app.py
1
1
2
3
5
8
13
21
34
55
```

For loop

The **for loop** in Python is used when we know the number of iterations or when we want to iterate over a sequence (like a list, string, or range).

Basic Syntax:

```
for variable in sequence:
    # Code to execute in each iteration
```

# Program to find the sum of squares of first n natural numbers

```
n = int(input('Enter the last number '))
sum = 0
for i in range(1, n+1):
    sum = sum + i*i                  #Explanation: 1²+2²+3²+4²+5²=1+4+9+16+25=55
print('The sum is ', sum)
```

Output:

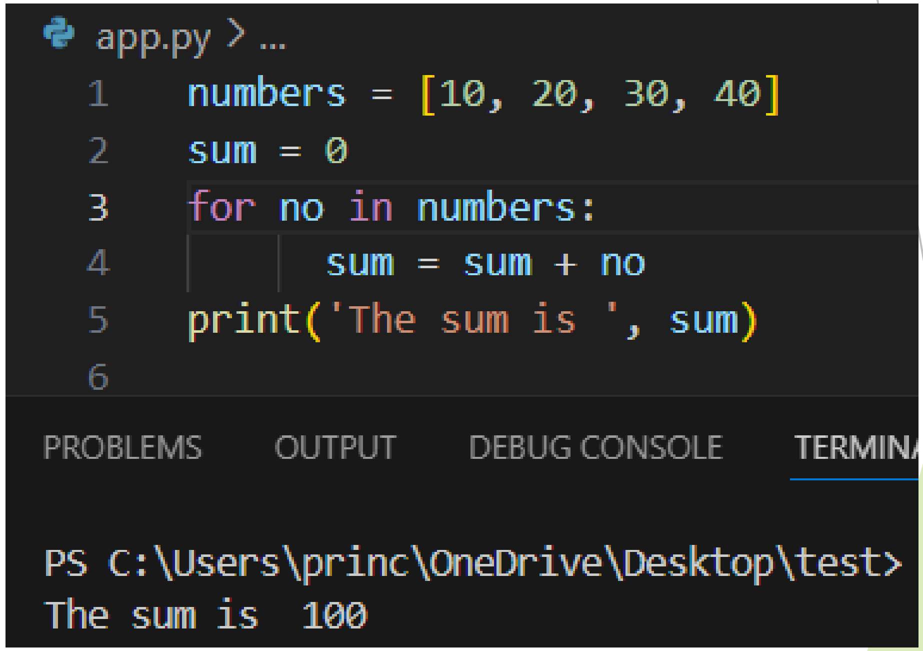
```
Enter the last number 5
The sum is 55
```

For loop

```
# Program to find the sum of a given set of numbers

numbers = [11, 17, 24, 65, 32, 69]

sum = 0

for no in numbers:

    sum = sum + no

print('The sum is ', sum)
```

Output:

```
The sum is 218
```



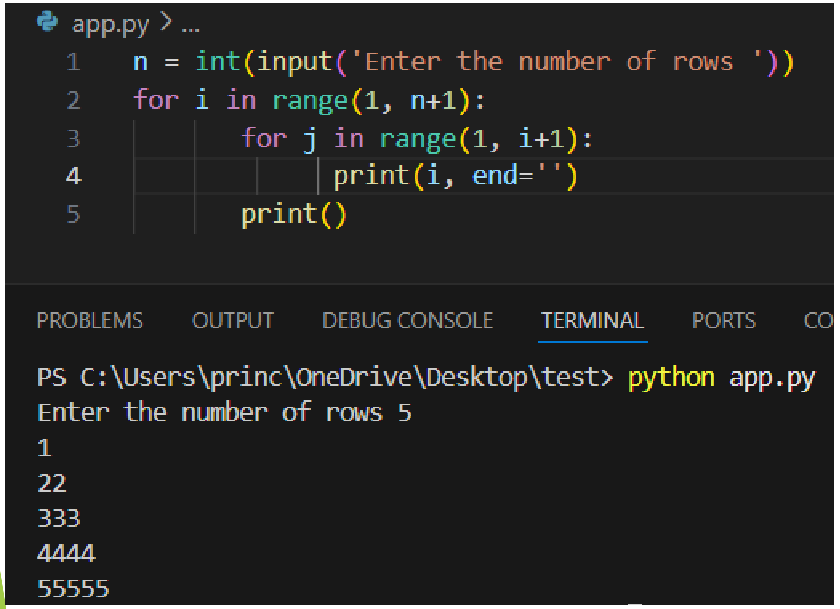
```
numbers = [10, 20, 30, 40]
sum = 0
for no in numbers:
    sum = sum + no
print('The sum is ', sum)
```

```
PS C:\Users\princ\OneDrive\Desktop\test>
The sum is  100
```

# Program to print 1, 22, 333, 444, ... in triangular form

```
for i in range(6):  # Outer loop: controls the rows (0 to 5)
    for j in range(1, i+1):  # Inner loop: controls the columns (1 to i)
        print(i, end='')  # Print 'i' without newline
    print()  # Move to the next line after inner loop completes
```

Output:

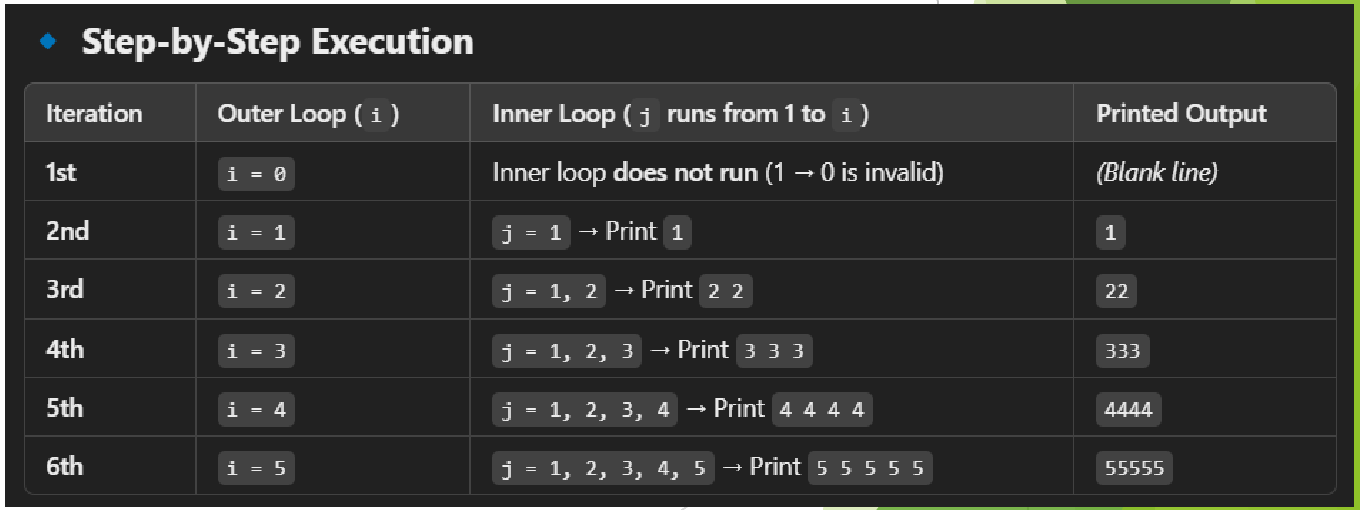


```
app.py > ...
n = int(input('Enter the number of rows '))
for i in range(1, n+1):
    for j in range(1, i+1):
        print(i, end='')
    print()
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   CO

```
PS C:\Users\princ\OneDrive\Desktop\test> python app.py
Enter the number of rows 5
1
22
333
4444
55555
```

```
1

22

333

4444

55555
```



Step-by-Step Execution

Iteration	Outer Loop (`i`)	Inner Loop (`j` runs from 1 to `i`)	Printed Output
1st	`i = 0`	Inner loop does not run (1 → 0 is invalid)	*(Blank line)*
2nd	`i = 1`	`j = 1` → Print `1`	`1`
3rd	`i = 2`	`j = 1, 2` → Print `2 2`	`22`
4th	`i = 3`	`j = 1, 2, 3` → Print `3 3 3`	`333`
5th	`i = 4`	`j = 1, 2, 3, 4` → Print `4 4 4 4`	`4444`
6th	`i = 5`	`j = 1, 2, 3, 4, 5` → Print `5 5 5 5 5`	`55555`

# Program to print opposite right triangle

```
n = int(input('Enter the number of rows '))
for i in range(n, 0, -1):
    for j in range(1, i+1):
        print('*', end='')
    print()
```

Output:

```
*****
****
***
**
*
```

# Program to print opposite star pattern

```
n = int(input('Enter the number of rows '))
for i in range(0, n):
    for j in range(0, n-i):
        print(' ', end='')
    for k in range(0, i+1):
        print('*’, end='')
    print('')
```

Output:

```
    *
   **
  ***
 ****
*****
```

```
# Program to print A,  AB,  ABC,  ABCD, ......
ch = str(input('Enter a character '))
val=ord(ch)
for i in range(65, val+1):
    for j in range(65, i+1):
        print(chr(j), end='')
    print()
```

Output:

```
A
AB
ABC
ABCD
ABCDE
ABCDEF
```

# Program to test Palindrome numbers

```
n=int(input('Enter an integer '))
x=n
r=0
while n>0:
    d=n%10
    r=r*10+d
    n=n//10
    if x==r:
           print(x,' is Palindrome number’)
    else:
          print(x, ' is not Palindrome number')
```

# Program to print Pascal Triangle

```
n=int(input('Enter number of rows '))
for i in range(0, n):
    for j in range(0, n-i-1):
        print(end=' ')
    for j in range(0, i+1):
        print('*', end=' ')
    print()
```

Output:

```
Enter number of rows 6
     *
    * *
   * * *
  * * * *
 * * * * *
* * * * * *
```

Break and Continue

In Python, break and continue statements can alter the flow of a normal loop.

```
# Searching for a given number

numbers = [11, 9, 88, 10, 90, 3, 19]

for num in numbers:

    if(num==88):

        print("The number 88 is found")

        break
```

Output:

```
The number 88 is found
```

Break and Continue

```
# program to display only odd numbers
for num in [20, 11, 9, 66, 4, 89, 44]:
    # Skipping the iteration when number is even
    if num%2 == 0:
        continue
        # This statement will be skipped for all even numbers
    else:
        print(num)
```

File

A file is some information or data which stays in the computer storage devices.

Files are of two types:

- text files
- binary files.

Text files:

We can create the text files by using the following syntax:

Variable name = open (“file.txt”, file mode)

Example:

f= open ("hello.txt","w+“)

File modes

Mode	Description
‘r’	This is the default mode. It Opens file for reading.
‘w’	This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file.
‘x’	Creates a new file. If file already exists, the operation fails.
‘a’	Open file in append mode. If file does not exist, it creates a new file.
‘t’	This is the default mode. It opens in text mode.
‘b’	This opens in binary mode.
‘+’	This will open a file for reading and writing (updating) .

Creating output file

```
file = open(‘output.txt’, ‘a+’)


items = [‘mango’, ‘orange’, ‘banana’, ‘apple’]
for item in items:
        print(item, file = file)
file.close()
```

# Write a python program to open and read a file

```
 a=open(“one.txt”, ”r”)
content = a.read()
print(content)
a.close( )
```

# Write a python program to open and write “hello world” into a file.

```
 f=open("file.txt","a")
f.write("hello world")
f.close( )
```

# Python program to write the content “Hi python programming” for the existing file.

```
 f=open("MyFile.txt",'w')
f.write("Hi python programming")
f.close()
```

# Write a python program to open and write the content to file and read it.

```
 f=open("abc.txt","w+")
f.write("Python Programming")
print(f.read())
f.close()
```

References

[1] Kenneth A Lambert, Fundamentals of Python: First programs, 2nd edition – Cengage Learning India, 2019.

[2] Saha Amit, Doing Math with Python - No starch press, San Francisco, 2015.

[3] E. Balgurusamy, Problem solving and Python programming- Tata McGraw Hill, 2017.

[4] Bill Lubanovic, Introducing Python, Shroff Publishers & Distributors Pvt. Ltd., 2nd Edition, 2020.

NIELIT

THANK YOU