Python Basics

→ Operators

1) Arithmetic operators
2) Assignment operators
3) Comparison operator
4) Logical operator
5) Bitwise operator

Note - precedence is same

→ Indentation - in python indentation is extremely important

→ Reading input and producing output -

```
variable_name = input(prompt)
```

Producing output - in Python output is produced in two ways - a) str.format()
b) f-strings

Method 1- Str.format() Method-

Syntax :- str.format(P0, P1, ... K0 = V0, K1 = V1...)

P0, P1, ... → Positional arguments

K0 = V0, K1 = V1... → Keyword arguments

Eg- (i) country = input("Which country do you live in?")
print("I live in {0}".format(country))

(ii) a = 10
b = 20
print("The values of a is {0} and b is {1}".format(a,b))
# interchange the positions
print("The value of b is {1} and a is {0}" format(a,b))

Method 2- F strings-

(i) country = input("Which country do you live in?")
print(f"I live in {country}")

→ Variable name

→ Type Conversions

- **The *int()* Function**
- To explicitly convert a float number or a string to an integer, cast the number using *int()* function.

The *float()* Function : returns a floating point number constructed from a number or string.

The *str()* Function : The *str()* function returns a string which is fairly human readable.

The *chr()* Function : Convert an integer to a string of one character whose ASCII code is same as the integer using *chr()* function. The integer value should be in the range of 0–255.

```
>>> char=chr(100)
>>> char          'd'
```

The *complex()* Function: to print a complex number with the value real + imag*j or convert a string or number to a complex number.

```
>>> complexstr=complex("2")   , >>> complexnum=complex(3,4)
>>> complexstr
(2+0j)                        >>> complexnum          (3+4j)
```

The *ord()* Function : returns an integer representing Unicode code point for the given Unicode character.

The *hex()* Function : Convert an integer number (of any size) to a lowercase hexadecimal string prefixed with "0x" using *hex()* function.

The *oct()* Function : Convert an integer number (of any size) to an octal string prefixed with "0o" using *oct()* function.

```
>>> hex1=hex(15)
>>> hex1
'0xf'
>>> hex1=hex(16)
>>> hex1
'0x10'
```

```
>>> oct1=oct(10)
>>> oct1
'0o12'
```

→ The type() Function and Is Operator

Syntax - type(object)

→ Control Flow Statements

Python Programming Language consists of

1. **Sequential Control Flow Statements:** line by line execution, in which the statements are executed sequentially, in the same order in which they appear in the program.

2. **Decision Control Flow Statements:** Depending on whether a condition is True or False, the decision structure may skip the execution of an entire block of statements or even execute one block of statements instead of other (if, if...else and if...elif...else).

3. **Loop Control Flow Statements:** This is a control structure that allows the execution of a block of statements multiple times until a loop termination condition is met (*for* loop and *while* loop). Loop Control Flow Statements are also called **Repetition statements or Iteration statements**.

1) If Statement :-
```
if (Boolean_expression):
    statement(s)
```

2) If else :-
```
if (B_e):
    statement 1
else:
    statement 2
```

3) If elif else :-
```
if (B_e):
    statement
elif B_e:
    statement
    .
    .
else:
```

Statement

4) Nested if -

```
if B_e:
    if B_e:
        statement_1
    else:
        statement_2
else:
    statement_3
```

5) While loop -

```
while B_e:
    Statements
```

6) For loop - syntax of range - range([start] stop [,step])

start → optional

step → optional

```
for iteration_variable in sequence:
    statements(s)
```

Note - continue and break statements work in the same way

(Pr) WAP to add two numbers.

(Sol)
```
num1 = input("Enter the first number:")
num2 = input("Enter the second number:")
sum = num1 + num2
print(f"The sum of two numbers is {sum}")
```

→ Loops in Python-

1.) While loop - declare variable
Set condition
increment / decrement value

2.) For loop- for var in range (start, stop, step):
↓
not counted

(Pr) WAP to add up all the even numbers using while loop.

(Sol)
```
num1 = int(input("Enter the first number:"))
num2 = int(input("Enter the second number:"))
sum = 0
while x<=y
```

```
sum += x
x = x+2
print(f"The sum is {sum}")
```

(Pr) WAP to print the factorial of a given number

(Sol)
```
num = int(input("Enter a number"));
fact = 1
num<0, num==0 (or) for i in range(1, num+1)
    fact = fact * i
```

(Pr) WAP to print 'n' natural numbers

(Sol)
```
n = int(input("Enter the limit:"))
for i in range(1, n+1)
    print(i)
```

(Pr)

10. Write a program to read the Richter magnitude value from the user and display the result for the following conditions using ***if...elif...else*** statement.

Richter Magnitude	Information
> 1.0 and < 2.0	Microearthquakes not felt or rarely felt
> 2.0 and < 4.0	Very rarely causes damage
> 4.0 and < 5.0	Damage done to weak buildings
> 5.0 and < 6.0	Cause damage to the poorly constructed building
> 6.0 and < 7.0	Causes damage to well-built structures
> 7.0 and < 8.0	Causes damage to most buildings
> 8.0 and < 9.0	Causes major destruction
> 9.0	Causes unbelievable damage

richter_magnitude = int(input("Enter the magnitude on the Richter scale:"))

if richter_magnitude > 1.0 && richter_magnitude < 2.0:
    print(" ")

elif ...

:

:

(Pr) Write a program to print the sum of the following series

a. $1 + ½ + 1/3 + \ldots + 1/n$

b. $1/1 + 2^2/2 + 3^3/3 + \ldots\ldots + n^n/n$

(Sol)

```
n = int(input("Enter the limit for calculation"))
sum = 0
for i in range(1, n):
    sum += 1/i
print("The sum is {sum}")
```

→ Use f strings

$1 + \frac{1}{2} = \frac{3}{2} = 1$

(Pr) WAP to calculate sum of natural no's

(Sol)

```
while (num>0)
```

sum += num

num -= 1

→ Functions -

- Built in functions such as abs(), min(), max(), divmod(), pow(), len()

A Few Built-in Functions in Python

Function Name	Syntax	Explanation
abs()	*abs(x)* where x is an integer or floating-point number.	The abs() function returns the absolute value of a number.
min()	*min(arg_1, arg_2, arg_3,…, arg_n)* where arg_1, arg_2, arg_3 are the arguments.	The min() function returns the smallest of two or more arguments.
max()	*max(arg_1, arg_2, arg_3,…,arg_n)* where arg_1, arg_2, arg_3 are the arguments.	The max() function returns the largest of two or more arguments.
divmod()	*divmod(a, b)* where a and b are numbers representing numerator and denominator.	The divmod() function takes two numbers as arguments and return a pair of numbers consisting of their quotient and remainder. For example, if a and b are integer values, then the result is the same as (a // b, a % b). If either a or b is a floating-point number, then the result is (q, a % b), where q is the whole number of the quotient.
pow()	*pow(x, y)* where x and y are numbers.	The pow(x, y) function returns x to the power y which is equivalent to using the power operator: x**y.
len()	*len(s)* where s may be a string, byte, list, tuple, range, dictionary or a set.	The len() function returns the length or the number of items in an object.

Syntax for function defined in a module is,

module_name.function_name()

Syntax for self created functions

```
def greet():
    print("Hi there")
    print("Welcome aboard")

greet()
```

Input :-

```
def greet(first_name, last_name):
    print(f"Hi {first_name} {last_name}")
    print("Welcome aboard")

greet("Arnav", "Karnik")
```

→ Types of functions

Returning a value:

```
def get_greeting(name):
    return f"Hi {name}"

message = get_greeting("Arnav")
print(message)
```

Syntax for defining a function

def function_name (parameter_1, parameter_2... parameter_n):

    Statements

```
def function_definition_with_no_argument():
    print("This is a function definition with NO Argument")


def function_definition_with_one_argument(message):
    print(f"This is a function definition with {message}")


def main():
    function_definition_with_no_argument()
    function_definition_with_one_argument("First Argument")


if __name__ == "__main__":
    main()
```

→ The return statement and void function

Syntax - return [expression_list]

```
]: def world_war():
       alliance_world_war = input("Which alliance won World War 2?")
       world_war_end_year = input("When did World War 2 end?")
       return alliance_world_war, world_war_end_year


   def main():
       alliance, war_end_year = world_war()
       print(f"The war was won by {alliance} and the war ended in {war_end_year}")


   if __name__ == "__main__":
       main()
```

```
Which alliance won World War 2?Allies
When did World War 2 end?1945
The war was won by Allies and the war ended in 1945
```

Two types of variables → Global variables
Local variables

→ *args and **kwargs

- used as parameters in functions
- allow you to pass a variable no of arguments to the calling function

]:
```python
# Program 4.11: Program to Demonstrate the Use of *args and **kwargs

def cheese_shop(kind, *args, **kwargs):
    print(f"Do you have any {kind} ?")
    print(f"I'm sorry, we're all out of {kind}")
    for arg in args:
        print(arg)
    print("-" * 40)
    for kw in kwargs:
        print(kw, ":", kwargs[kw])


def main():
    cheese_shop("Limburger", "It's very runny, sir.",
                "It's really very, VERY runny, sir.",
                shop_keeper="Michael Palin",
                client="John Cleese",
                sketch="Cheese Shop Sketch")


if __name__ == "__main__":
    main()
```

(Pr) WAP to print largest of 3 no's function

(Sol)

```
def max_of_three(num1, num2, num3):
    if num1 > num2 and num1 > num3:
        return num1
    elif num2 > num1 and num2 > num3:
        return num2
    else:
        return num3
def main():
    num1 = int(input("Enter the first number:"))
```

```
num2 = int(input("Enter the second number:"))
num3 = int(input("Enter the third number:"))
maximum = Max_of_three(num1, num2, num3)
print(f"The greatest of the three is {maximum}.")
if __name__ == "__main__":
    main()
```

(Pr) WAF for HS → Harmonic Series.

(Sol)

$$\sum_{n=1}^{\infty} \frac{1}{n} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} \cdots \frac{1}{n}$$

$$1 + \frac{1}{2} =$$

return sum

everything else is the same

→ Strings-

- A sequence of characters including letters, numbers, punctuation marks and spaces.

- Another data type

→ Str() Function

- used to return a string

- if object is not provided, it returns an empty string

→ Basic String operations

- to concatenate strings we use '+' operator

- to create a repeated sequence of strings - '*' operator

(Pr) WAP to concatenate 2 strings.

(Sol)
```
String 1 = str(input("Enter string 1"))
String2 = str(input("Enter String 2"))
Strsum = string1 + string2
```

(Pr) WAP to print a string n times

(Sol)
```
Strprod = string * n
```

→ String comparison

- Can use >, <, <=, >=, ==, != to compare 2 strings which will either return a boolean true or a boolean false value.

Built in functions :- len() - length of string
max() - highest ASCII value
min() - lowest ASCII value

To access characters in string by Index Number-

String_name [index number]

→ String slicing and joining

What is string slicing ?
- refer to sub-parts of sequence of characters

Syntax - String_name[start:end [:step]]

What is string joining?

- refers to joining strings

Syntax - string_name.join(sequence)

→ Split strings using split() Method-

- returns a list of string items by breaking up the string

Syntax- string_name.split([separator [, maxsplit]]

delimiter string (optional)

→ String method-

1) conversion purposes- capitalize(), lower(), upper(), swapcase(), title() and count()

2) comparing purposes- islower(), isupper(), isdecimal(), isdigit(), isnumeric(), isalpha(), isalnum

3.) find() → used to find a substring in a string

4.) replace(), join(), split(), splitlines() used to replace a string in python:

Pr) WAP to print the characters that are common:

Sol)
```
string1 = str(input("Enter the first string"))
string2 = str(input("Enter the second string"))
if(string1 != string2):
    print(" cannot find common characters
    else
     string1[:] == string2[:]
```

→ Lists

- list is a container that holds a number of items

↳ 1D Array

Syntax- list_name = [item_1, item_2 .... item_n]

→ Basic Lists operation

- '+' and '*' operator can also be used to create a repeated sequence of list items

- list() function → used to create a list

→ Indexing and Slicing in lists -

- To access an item in a list ⇒ list_name[index]

Modifying items in lists :- list1[-2] = 'Sun'

lst[1]

['mon', 'tue', 'Sun', 'Thu']

Slicing operation - extracts a part of the list

Syntax- list_name [start : stop [: step]]

→ Built in Functions used on lists-

Built-In Functions Used on Lists

Built-In Functions	Description
len()	The *len()* function returns the numbers of items in a list.
sum()	The *sum()* function returns the sum of numbers in the list.
any()	The *any()* function returns *True* if any of the Boolean values in the list is *True*.
all()	The *all()* function returns *True* if all the Boolean values in the list are *True*, else returns *False*.
sorted()	The *sorted()* function returns a modified copy of the list while leaving the original list untouched.

→ List methods-

List Methods

Various List Methods

List Methods	Syntax	Description
append()	list.append(item)	The *append()* method adds a single item to the end of the list. This method does not return new list and it just modifies the original.
count()	list.count(item)	The *count()* method counts the number of times the item has occurred in the list and returns it.
insert()	list.insert(index, item)	The *insert()* method inserts the item at the given index, shifting items to the right.
extend()	list.extend(list2)	The *extend()* method adds the items in list2 to the end of the list.
index()	list.index(item)	The *index()* method searches for the given item from the start of the list and returns its index. If the value appears more than once, you will get the index of the first one. If the item is not present in the list then *ValueError* is thrown by this method.
remove()	list.remove(item)	The *remove()* method searches for the first instance of the given item in the list and removes it. If the item is not present in the list then *ValueError* is thrown by this method.
sort()	list.sort()	The *sort()* method sorts the items *in place* in the list. This method modifies the original list and it does not return a new list.
reverse()	list.reverse()	The *reverse()* method reverses the items *in place* in the list. This method modifies the original list and it does not return a new list.
pop()	list.pop([index])	The *pop()* method removes and returns the item at the given index. This method returns the rightmost item if the index is omitted.

→ Nested Lists -

Syntax- nested_list_name = [[item_1, item_2, item_3],
[item_4, item 5, item 6],
[item_7, item_8, item9]]

→ The del Statement - del statement

7. Write a program that creates a list of 10 random integers. Then create two lists by name odd_list and even_list that have all odd and even values of the list respectively.
8. Write a program to sort the elements in ascending order using insertion sort.
9. Write a Python program to use binary search to find the key element in the list.
10. Make a list of the first eight letters of the alphabet, then using the slice operation do the following operations:
    a. Print the first three letters of the alphabet.
    b. Print any three letters from the middle.
    c. Print the letters from any particular index to the end of the list.
11. Write a program to sort the elements in ascending order using selection sort.
12. Write a program that prints the maximum value of the second half of the list.
13. Write a program that creates a list of numbers 1–100 that are either divisible by 5 or 6.
14. Write a function that prompts the user to enter five numbers, then invoke a function to find the GCD of these numbers.

(7Sol) - function definition:-

```
def odd (listodd):
    print ("The odd list is")
    for i in range(len(listodd)):
        if listodd[i]%2 != 0:
            Print (listodd[i])
```

// same for even //

Main -

```
def main()
    input_string = input ("Enter integer in
                          the given...")
list main = [int(x) for x in input_string.split()]
  odd (listmain)
  even (listmain)
```

↓

// split the string //

Note - When you input a list of numbers, you should split the input string and convert each element to an integer

list_man = [int(x) for x in input_str.split()]

→ Dictionaries -

- Consists of key value pairs ordered and changeable
  Eg- item : price

capitals = {"USA": "Washington DC", "India": "New Delhi"}

dir(capitals) → function of dictionary

Syntax - dictionary_name = {key_1 : value_1, key_2 : value_2 ...}

Note - No duplicates are allowed

→ Accessing and modifying key : value Pairs in Dictionaries

To access value :- dictionary_name[key]

To modify :- dictionary_name[key] = value

To check if a key is present use in and not in membership operators

↳ returns either a boolean true (or) false value

Built-In Functions Used on Dictionaries

Built-in Functions	Description
len()	The *len()* function returns the number of items (*key:value* pairs) in a dictionary.
all()	The *all()* function returns Boolean True value if all the keys in the dictionary are True else returns False.
any()	The *any()* function returns Boolean True value if any of the key in the dictionary is True else returns False.
sorted()	The *sorted()* function by default returns a list of items, which are sorted based on dictionary keys.

Various Dictionary Methods

Dictionary Methods	Syntax	Description
clear()	dictionary_name.clear()	The *clear()* method removes all the *key:value* pairs from the dictionary.
fromkeys()	dictionary_name.fromkeys(seq [, value])	The *fromkeys()* method creates a new dictionary from the given sequen elements with a value provided by the user.
get()	dictionary_name.get(key [, default])	The *get()* method returns the value associated with the specified key in t dictionary. If the key is not present then it returns the default value. If c is not given, it defaults to *None*, so that this method never raises a *KeyE*
items()	dictionary_name.items()	The *items()* method returns a new view of dictionary's key and value p as tuples.
keys()	dictionary_name.keys()	The *keys()* method returns a new view consisting of all the keys in the dictionary.
pop()	dictionary_name.pop(key[, default])	The *pop()* method removes the key from the dictionary and returns its valu the key is not present, then it returns the default value. If default is not gi and the key is not in the dictionary, then it results in *KeyError*.
popitem()	dictionary_name.popitem()	The *popitem()* method removes and returns an arbitrary (key, value) tu pair from the dictionary. If the dictionary is empty, then calling *popite* results in *KeyError*.
setdefault()	dictionary_name.setdefault (key[, default])	The *setdefault()* method returns a value for the key present in the dictio If the key is not present, then insert the key into the dictionary with a default value and return the default value. If key is present, *default* defaults to None, so that this method never raises a *KeyError*.
update()	dictionary_name.update([other])	The *update()* method updates the dictionary with the *key:value* pairs fro *other* dictionary object and it returns *None*.
values()	dictionary_name.values()	The *values()* method returns a new view consisting of all the values in dictionary.

Note: Replace the word *"dictionary_name"* mentioned in the syntax with your *actual dictionary name* in your

→ update() → used to update the key values of the dictionary

→ to delete used del dictname[key]

How to build a basic dictionary ?

```
def main()
    dictionary_name = {}
    n= int(input("Enter the number of inputs that are required"))
    for i in range(0,n)
        dict_key = input("Enter key")
        dict_value= input("Enter value")
    dictionary_name.update({dict_key: dict_value})
    print(dictionary_name)
if __name__ == "__main__"
    main()
```

→ Tuples-

List	Tuple
Mutable	Immutable
↓	↓
Can change	Can't change after inserting

tuple_name = (elements ...)

a = (1, 2, 3, 4, 5)

a = (1) ✗ → Not allowed

a = (1,) ✓ → allowed

} creating a tuple

a = (1, 2, 3, 4)

b = (5, 6, 7, 8)

c = a + b

(1, 2, 3, 4, 5, 6, 7, 8)

updating a tuple

del a - delete the entire tuple

Slicing → giving index range → used to access elements

a = [1, 2, 3, 4] → To slice a[0:2]
- (1, 2, 3)

a[0] → 1

[:]

a = (10, 20, 30, 40)
b = (50, 60, 70, 80)

- Basic operations on tuples -

1) Length - len(a) = 40

2) Concatenation (+) ⇒ a+b = (10, 20, 30, 40, 50, 60, 70, 80)

3) Repetition (*) → repeats

4) Membership (in/not in) → 10 in a ⇒ true

5) Iterative statements → for i in a → 10, 20, 30, 40
print(i) ↳ list

6) Maximum → max(a)

7) Minimum → min(a)

8) Index → a.index(2) = 30

9) count → a.count(20) = 1

10) Conversion to tuple - tuple( )

→ Sets-

S= {1, 'a', 2.5, "abc"}

- Avoids duplications

S= {} → creating empty set

S.add(1) → {1}

S.add(2) → {1,2}

S.add(3) → {1,2,3}

S= {1,2,3}
t= {4,5,6}
S.update(t)
print(s) → {1,2,3,4,5,6}

t.update(s)

print(t) → {4, 5, 6, 1, 2, 3}

- Basic operations on sets -

s = {1, 2, 3}

t = {3, 4, 5}

1.) s.add(6) → s = {1, 2, 3, 6}

2.) len(s) = 4

3.) s.update(t) → {1, 2, 3, 6, 4, 5}

4.) s.remove(6) → {1, 2, 3, 4, 5}

5.) s.discard(5) → {1, 2, 3, 4}

6.) s.pop() → 1 FIFO

7.) s.clear() → s = {}

Can't perform indexing and slicing

$S = \{1, 3, 2, 4, 6, 5\}$

1.) max(S) = 6

2.) min(S) = 1

3.) sum(S) = 21

$S = \{1, 2, 3\}$

$t = \{1, 2, 3, 4, 5, 6\}$

1.) issubset()

S.issubset(t) → TRUE

2.) issuperset()

S.issuperset(t) → FALSE

3.) S.union(t) → $\{1, 2, 3, 4, 5, 6\}$

4) s. intersection (t) → {1, 2, 3}

5) s. difference (t) → All the elements of s except common will be printed

For frozenset use frozen() → will be immutable

→ Files-

File - collection of data

open()

Syntax - file_object = open ("filename.txt", "Access Mode")

↳ file pointer

↳ points to the content of the file

Access modes

r- read mode
- only for reading the content
- file pointer points at the beginning of the file
- file should exist before opening in read mode

```
fp= open("abc.text", "r")
```

w- write mode
- only for writing the content
- File pointer points at the beginning of the file

① If the file exists the file is opened and pointer points at beginning

② If file doesn't exist then the new file will be created with the given file name

```
fp = ("abc.txt", "w")
```

a - append mode

- only to append and write the data
- file pointer points at the end of the content file

```
fp= ("abc.txt", "a")
```

r+ ⇒ Reading and writing

w+ ⇒ Reading and writing

a+ ⇒ Reading and appending

To close the file ⇒ file_object.close()

rb

wb

ab

rb+

wb+

ab+

} Binary files

- Reading and Writing operations on files -

```
f1 = open("abc.txt", "w")
```

write()

```
f1.write(string)
```

Eg -

```
f1 = open("abc.txt", w)
f1.write("Welcome to Python")
```

writelines()

```
f1.writelines(list)
```

Eg -

```
F1 = open("abc.txt", "w")
lines = ["hello", "welcome", "Python programming"]
f1.writelines(lines)
```

Reading

```
f1 = open("abc.txt", "r")
f1.read(bit positions)
f1.read(5)
```

abc.txt

↳ Hello

f1.read(5)

↳ -Worl

f1.read() → Hello world

readline() → one line at a time

writeline() → one line at a time

readlines() → all lines

writelines() → all lines