BASIC INPUT AND OUTPUT STATEMENTS

- Output Statements
- Input Statements

OUTPUT STATEMENTS

- Provides the print() function.

The print() Statement

- It will throw the cursor to the next line.
- Blank line will be displayed.

The print("string") Statement

- The string is displayed as it is.
- When we use **'+'** on strings, it will join one string with another string.
- **'+'** is called **concatenation operator**.
- **',** comma used with print indicates that the values are different and observe the single space after the string.

Example:

```
print("Hello")

print('Hello')

#Escape sequence

print("This is the \nPython lecture")

print("This is the \tPython lecture")

#to escape the effect of escape sequence

print("This is the \\nPython lecture")

#repetition operator

print(3*'Hello')

# '+' it will join one string to another string

print("My name="+"Kriti")

print("My name=","Kriti")
```

The print(variable list) Statement

Example:

```
a,b = 2,8

print(a)

print(a,b)
```

- To separate the output with a comma, we should use **'sep'** attribute.
- **'sep'** represents separator.
- The format is sep =" characters" which are use to separate the values.

```
print(a,b, sep=",")
print(a,b,sep=":")
print(a,b,sep="------")
```

- Print() function not to throw the cursor into to next line but display the output in the same line./thus is done using **'end' attribute.** (end =" characters")

```
print("Hello")
print("Good Morning")
print("This is Python Session")

print("Hello", end=' ')
print("Good Morning", end=' ')
print("This is Python Session", end= '')

print("Hello", end='\t')
print("Good Morning", end='\t')
print("This is Python Session", end='\t')
```

The print(object) Statement

- We can pass objects like lists, tuples, dictionaries to display the elements of those objects.

Example:

```
lst=[67,'A','Hello'] #list
print(lst)

d={'a':10, 'b':20, 'c':30}  #dictionary

print(d)
```

The print(“string”, variable list) Statement

Example:

```
a=17
print(a,"Even number")
print('You typed',a,'as input')
```

The print(formatted string) Statement

- The special operator **‘%’ (percent)** can be used.
- It joins a string with a variable or value.

Syntax: print(“formatted string”%(variable list))

Example:

```
#To display a single variable
x=10
print('Value=%i' % x)
# To display more than one variable
x, y=10, 57
print('x=%i y=%d' % (x,y))

# To display a string
name='Priya'
print('Hello %s' % name)
print('Hello (%20s)' %name)
print('Hello (%-20s)' %name)

name='pooja'
print('Hello %c, %c' % (name[0], name[1]))  # To display single character
print('Hello %s' % (name[0:2]))    #slicing operator
```

 # To display Floating Numbers

```
num=567.89

print('the value is :%f' %num)

print('the value is :%5.2f' %num)

print('the value is :%.3f' %num)
```

- **Replacement field** which is denoted by a pair by curly braces {}.
- Names and indexes represent the order of the values.
- After the formatted string, we mention the values to be displayed

Syntax: print('format string with replacement fields'.format(values))

Example:

```
a1, a2, a3 = 1, 2, 3

print('number1={0}'.format(a1)) #{0} was replaced by the value of a1

print('number1={0}, number2={1}, number3={2}'.format(a1,a2,a3))

#Use names in replacement field

print('number1={two}, number2={one}, number3={three}'.format(one=a1,
two=a2, three=a3))

#Without mentioning indexes and names

print('number1={},number2={}, number3={}'.format(a1,a2,a3))
```

Example:

```
name, salary = 'Rajesh', 30000.75

print('Hello {0}, your salary is {1}'.format(name,salary))

print('Hello {n}, your salary is {s}'.format(n=name,s=salary))

print('Hello %s, your salary is %.2f' % (name,salary))
```

INPUT STATEMENTS

- Input() function.
- Takes a value from the keyboard and returns it as a string.

Example:

```
str=input('Enter your name:')
print(str)

str=input('Enter a number:')
x=int(str) #str is converted into int
print(x)

#For the integer values
x=int(input('Enter a number:'))
print(x)

#For the float values
x=float(input('Enter a number:'))
print(x)

#To accept a character as a string
ch=input("Enter a character:")
print("You entered:"+ch)

#index for single character
ch=input("Enter a character:")
print("You entered:"+ch[0])
```

- To accept more than one input in the same line, we can use a for loop along with the input() function in the following format:

a, b = [int(x) for x in input(“Enter two numbers:”).split()]

- Strings are divided wherever a space is found by split() method.
- [] indicated that the input is accepted as elements of a list.

Example:

```
a1, a2, a3=[int(x) for x in input("Enter three numbers:").split()]
print('Sum = ', a1+a2+a3)
```

- **Eval() function** is used to evaluate the strings.

Example:

```
a, b=5,10
result=eval("a+b-2")
print(result)
#using eval() along with input
x=eval(input("Enter the expression"))
print("Results=%d" %x)

#using eval() along with input and list
lst=eval(input("Enter the list"))
print("List= ", lst)
```

Command Line Arguments

Example: add.py

```
x=int(input('Enter first number:'))
y=int(input('Enter second number:'))
print('Sum is=',x+y)

python add.py
Enter first number:12
Enter second number:3
Sum is= 15
```

- Arguments are stored by default in the form of strings in a list with the name **'argv'** which is available in **sys module.**

  Argv[0] : represents the name of the program

  Argv[1]: represents the first value

  Argv[2]: represents the second value and so on.

- **Len() function: no. of command line arguments**

  Example:

```
# To Display command line args
```

```
import sys
n=len(sys.argv)  # n is the number of arguments
args = sys.argv # args list contains arguments
print('Number of command line args=',n)
print('The args are:',args)

print('The args one by one:')
for i in args:
        print(i)
```

➔ Python cmd1.py (On Command Prompt)

- ‘ “ Pooja Singh” ’ or “ ‘ Pooja Singh’ “ # as a whole argument

- Int(sys.argv[1]) #converts argv[1] into int type

- Float (sys.argv[2] # converts argv[2] into float type