CHAPTER 5

INPUT AND OUTPUT

The purpose of a computer is to process data and return results. It means that first of all, we should provide data to the computer. The data given to the computer is called *input*. The results returned by the computer are called *output*. So, we can say that a computer takes input, processes that input and produces the output, as shown in Figure 5.1:

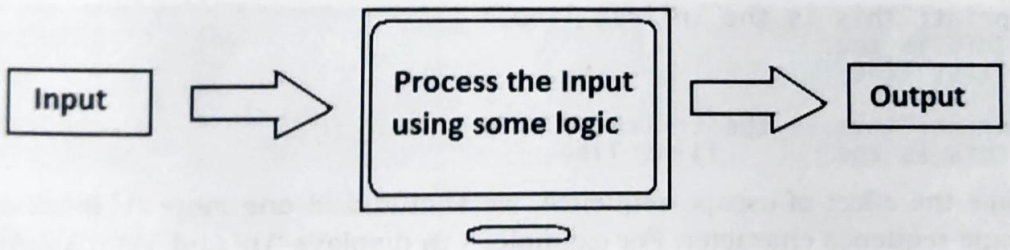

Figure 5.1: Processing Input by the Computer

To provide input to a computer, Python provides some statements which are called Input statements. Similarly, to display the output, there are Output statements available in Python. We should use some logic to convert the input into output. This logic is implemented in the form of several topics in subsequent chapters. We will discuss the input and output statements in this chapter, but in the following order:

- Output statements
- Input statements

Output statements

To display output or results, Python provides the print() function. This function can be used in different formats which are discussed hereunder.

The print() Statement

When the print() function is called simply, it will throw the cursor to the next line. It means that a blank line will be displayed.

The print("string") Statement

A string represents a group of characters. When a string is passed to the print() function, the string is displayed as it is. See the example:

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

Please remember that in case of strings, double quotes and single quotes have the same meaning and hence can be used interchangeably.

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

We can use escape sequence characters inside the print() function. An escape sequence is a character that contains a special meaning. For example, '\n' indicates new line. '\t' represents tab space.

```
print("This is the \nfirst line")
This is the
first line

print("This is the \tfirst line")
This is the       first line
```

To escape the effect of escape sequence, we should add one more '\' (backslash) before the escape sequence character. For example, \\n displays '\n' and \\t displays '\t'.

```
print("this is the \\nfirst line")
this is the \nfirst line
```

We can use repetition operator (*) to repeat the strings in the output as:

```
print(3*'Hai')
HaiHaiHai
```

The operator '+' when used on numbers will perform addition operation. The same '+' will not do addition operation on strings as it is not possible to perform arithmetic operations on strings. When we use '+' on strings, it will join one string with another string. Hence '+' is called concatenation (joining) operator when used on strings.

```
print("City name="+"Hyderabad")
City name=Hyderabad
```

The '+' operator in the preceding statement joined the two strings without any space in between. We can also write the preceding statement by separating the strings using ',' as:

```
print("City name=","Hyderabad")
City name= Hyderabad
```

In this case, a space is used by the print() function after each string. In the output, observe the single space after the string 'City name='. When the print() function sees a comma, it will assume that the values are different and hence a space should be used between them for clarity.

The print(variables list) Statement

We can also display the values of variables using the print() function. A list of variables can be supplied to the print() function as:

```
a, b = 2, 4
print(a)
2

print(a, b)
2 4
```

Default Separator is space.

Observe that the values in the output are separated by a space by default. To separate the output with a comma, we should use 'sep' attribute as shown below. 'sep' represents separator. The format is sep="characters" which are used to separate the values in the output.

```
print(a, b, sep=",")
2,4

print(a, b, sep=':')
2:4

print(a, b, sep='----')
2----4
```

When several print() functions are used to display output, each print() function will display the output in a separate line as shown below:

```
print("Hello")
print("Dear")
print('How are U?')
```

Output:

```
Hello
Dear
How are U?
```

Each print() function throws the cursor into the next line after displaying the output. This is the reason we got the output in 3 lines. We can ask the print() function not to throw the cursor into the next line but display the output in the same line. This is done using

'end' attribute. The way one can use it is end="characters" which indicates the ending characters for the line. Suppose, we write end='', then the ending character for each line will be '' (nothing) and hence it will display the next output in the same line. See the example:

```
print("Hello", end='')
print("Dear", end='')
print('How are U?', end='')
```

Output:

```
HelloDearHow are U?
```

If we use end='\t' then the output will be displayed in the same line but tab space will separate them as:

```
print("Hello", end='\t')
print("Dear", end='\t')
print('How are U?', end='\t')
```

Output:

```
Hello	Dear	How are U?
```

If we use end='\n' then the output is displayed in a separate line. So, '\n' is the default value for 'end' attribute.

The print(object) Statement

We can pass objects like lists, tuples or dictionaries to the print() function to display the elements of those objects. For example,

```
lst = [10, 'A', 'Hai']
print(lst)
[10, 'A', 'Hai']

d = {'Idly':30.00, 'Roti':45.00, 'Chappati':55.50}
print(d)
{'Idly': 30.0, 'Roti': 45.0, 'Chappati': 55.5}
```

The print("string", variables list) Statement

The most common use of the print() function is to use strings along with variables inside the print() function.

```
a=2
print(a, "is even number")
2 is even number

print('You typed ', a, 'as input')
You typed  2 as input
```

The print(formatted string) Statement

The output displayed by the print() function can be formatted as we like. The special operator '%' (percent) can be used for this purpose. It joins a string with a variable or value in the following format:

```
print("formatted string" % (variables list))
```

In the "formatted string", we can use %i or %d to represent decimal integer numbers. We can use %f to represent float values. Similarly, we can use %s to represent strings. See the example below:

```
x=10
print('value= %i' %  x)
value= 10
```

As seen above, to display a single variable (i.e. 'x') , we need not wrap it inside parentheses. When more than one variable is to be displayed, then parentheses are needed as:

```
x, y = 10, 15
print('x= %i y= %d' % (x, y))
x= 10 y= 15
```

To display a string, we can use %s in the formatted string. When we use %20s, it will allot 20 spaces and the string is displayed right aligned in those spaces. To align the string towards left side in the spaces, we can use %-20s. Consider the following examples:

```
name='Linda'
print('Hai %s' % name)
Hai Linda

print('Hai (%20s)' % name)
Hai (               Linda)

print('Hai (%-20s)' % name)
Hai (Linda               )
```

We can use %c to display a single character as shown below:

```
name='Linda'
print('Hai %c, %c' % (name[0], name[1]))
Hai L, i
```

The above example displayed 0th and 1st characters from name variable. We can use slicing operator on a string to display required characters from the string. For example, name[0:2] gives 0th to 1st characters from name as:

```
print('Hai %s' %(name[0:2]))
Hai Li
```

To display floating point values, we can use %f in the formatted string. If we use %8.2f, then the float value is displayed in 8 spaces and within these spaces, a decimal point and next 2 fraction digits.

When we use %.3f, then the float value is displayed with 3 fraction digits after the decimal point. However, the digits before decimal point will be displayed as they are. Consider the following examples:

```
num=123.456789
print('The value is: %f' % num)
The value is: 123.456789

print('The value is: %8.2f' %num)
```

```
The value is:   123.46   # observe 2 spaces before the value
print('The value is: %.2f' %num)
The value is: 123.46
```

Inside the formatted string, we can use replacement field which is denoted by a pair of curly braces { }. We can mention names or indexes in these replacement fields. These names or indexes represent the order of the values. After the formatted string, we should write member operator and then format() method where we should mention the values to be displayed. Consider the general format given below:

```
print('format string with replacement fields'.format(values))
```

To display a single value using index in the replacement field, we can write as:

```
n1, n2, n3 = 1, 2, 3
print('number1={0}'.format(n1))
number1=1
```

In the above statement, observe {0} which was replaced by the value of n1. To display all the three numbers, we can use:

```
print('number1={0}, number2={1}, number3={2}'.format(n1, n2, n3))
number1=1, number2=2, number3=3
```

In the above statement, {0}, {1}, {2} represent the values of n1, n2 and n3, respectively. Hence, if we change the order of these fields, we will have order of the values to be changed in the output as:

```
print('number1={1}, number2={0}, number3={2}'.format(n1, n2, n3))
number1=2, number2=1, number3=3
```

As an alternate, we can also use names in the replacement fields. But the values for these names should be provided in the format() method. Consider the following example:

```
print('number1={two}, number2={one}, number3={three}'.format(one=n1,
      two=n2, three=n3))
number1=2, number2=1, number3=3
```

We can also use the curly braces without mentioning indexes or names. In this case, those braces will assume the sequence of values as they are given. Consider the following statements:

```
print('number1={}, number2={}, number3={}'. format(n1, n2, n3))
number1=1, number2=2, number3=3
```

All the four examples given below will display the same output:

```
name, salary = 'Ravi', 12500.75
print('Hello {0}, your salary is {1}'.format(name, salary))
Hello Ravi, your salary is 12500.75

print('Hello {n}, your salary is {s}'.format(n=name, s=salary))
Hello Ravi, your salary is 12500.75

print('Hello {:s}, your salary is {:.2f}'.format(name, salary))
Hello Ravi, your salary is 12500.75

print('Hello %s, your salary is %.2f' % (name, salary))
Hello Ravi, your salary is 12500.75
```

Input Statements

To accept input from keyboard, Python provides the input() function. This function takes a value from the keyboard and returns it as a string. For example,

```
str = input()   # this will wait till we enter a string
Raj kumar  # enter this string

print(str)
Raj kumar
```

It is a better idea to display a message to the user so that the user understands what to enter. This can be done by writing a message inside the input() function as:

```
str = input('Enter your name: ')
Enter your name: Raj kumar

print(str)
Raj kumar
```

Once the value comes into the variable 'str', it can be converted into 'int' or 'float' etc. This is useful to accept numbers as:

```
str = input('Enter a number: ')
Enter a number: 125
x =  int(str)  # str is converted into int
print(x)
125
```

We can use the int() function before the input() function to accept an integer from the keyboard as:

```
x = int(input('Enter a number: '))
Enter a number: 125
print(x)
125
```

Similarly, to accept a float value from the keyboard, we can use the float() function along with the input() function as:

```
x = float(input('Enter a number: '))
Enter a number: 12.345
print(x)
12.345
```

We will understand these concepts with the help of a Python program. Let's write a program to accept a string and display it, as shown in Program 1.

Program

Program 1: A Python program to accept a string from keyboard and display it.

```
# accepting a string from keyboard
str = input("Enter a string: ")
print('U entered: ', str)  #display entire string
```

Output:

```
C:\>python Input.py
Enter a string: Hello
U entered: Hello
```

The way we accept a character from the keyboard is also same as accepting a string. This is shown in Program 2.

Program

Program 2: A Python program to accept a character as a string.

```
# accepting a single character or string from keyboard
ch = input("Enter a char: ")
print("U entered: "+ch)
```

Output:

```
C:\>python Input.py
Enter a char: A
U entered: A

C:\>python Input.py
Enter a char: abcd
U entered: abcd
```

From the output of the above program, we can understand that the input() function is accepting the character as a string only. If we need only a character, then we should use the index, as: ch[0]. Here 0th character is taken from the string. This is shown in Program 3.

Program

Program 3: A Python program to accept a single character from keyboard.

```
# accepting a single character from keyboard
ch = input("Enter a char: ")
print("U entered: "+ch[0])
```

Output:

```
C:\>python Input.py
Enter a char: abcd
U entered: a
```

Now, let's plan a program to accept an integer number from the keyboard. Since, the input() function accepts only a string, it will accept the integer number also as a string. Hence, we should convert that string into integer number using the int() function. This is shown in Program 4.

Program

Program 4: A Python program to accept an integer number from keyboard.

```
# accepting integer from keyboard
str = input('Enter a number: ')
```

```
x = int(str)  #convert string into int
print('U entered: ', x);  #display the int number
```

Output:

```
C:\>python Input.py
Enter a number: 88778
U entered: 88778
```

The same program can be rewritten by combining the first two statements, as shown in Program 5.

Program

Program 5: A Python program to accept an integer number from keyboard – version 2.

```
# accepting integer number from keyboard - v2.0
x = int(input('Enter a number: '))
print('U entered: ', x);  #display the int number
```

Output:

```
C:\>python Input.py
Enter a number: 98798798423422
U entered: 98798798423422
```

The same procedure can be adopted to accept a floating point number from the keyboard. While the input() function accepts a float value as a string, it should be converted into float number using the float() function. This is shown in Program 6.

Program

Program 6: A Python program to accept a float number from keyboard.

```
# accepting float number from keyboard
x = float(input('Enter a number: '))
print('U entered: ', x);  #display the float number
```

Output:

```
C:\>python Input.py
Enter a number: 9.123456789123456789
U entered: 9.123456789123457
```

From the above output, we can understand that the float values are displayed to an accuracy of 15 digits after decimal point. In the next program, we will accept two integer numbers and display them.

Program

Program 7: A Python program to accept two integer numbers from keyboard.

```
# accepting two numbers from keyboard
x = int(input('Enter first number: '))
y = int(input('Enter second number: '))
print('U entered: ', x, y) #display both the numbers separating with a space
```

Output:

```
C:\>python Input.py
Enter first number: 12
Enter second number: 45
U entered: 12 45
```

The two numbers in the output are displayed using a space. Suppose we want to display these numbers using a comma as separator, then we can use sep=',' in the print() function as:

```
print('U entered: ', x, y, sep=',');
```

In this case, the output will be:

```
U entered: ,12,45
```

Observe the commas after 'U entered:' and after 12. A better way to display these numbers separating them using commas is this:

```
print('U entered: %d, %d' %(x, y))
```

Let's write a Python program to accept two numbers from keyboard and find their sum. This is shown in Program 8.

Program

Program 8: A Python program to accept two numbers and find their sum.

```
# find sum of two numbers
x = int(input('Enter first number: '))
y = int(input('Enter second number: '))
print('The sum of ',x, ' and ', y,' is ', x+y)  #display sum
print('The sum of {} and {} is {}'.format(x,y, x+y))  #display again
```

Output:

```
C:\>python Input.py
Enter first number: 88
Enter second number: 98
The sum of  88  and  98  is  186
The sum of 88 and 98 is 186
```

Let's find the product of two given numbers. The previous program is improved to find product of the numbers as shown in Program 9.

Program

Program 9: A Python program to find sum and product of two numbers.

```
# find sum and product of two numbers
x = int(input('Enter first number: '))
y = int(input('Enter second number: '))

#display sum
print('The sum of {0} and {1} is {2}'.format(x,y, x+y))
#display product
print('The product of {0} and {1} is {2}'.format(x,y, x*y))
#display both sum and product
```

```
print('The sum of {0} and {1} is {2} and product of {0} and {1} is
      {3}'.format(x,y, x+y, x*y))
```

Output:

```
C:\>python Input.py
Enter first number: 45
Enter second number: 65
The sum of 45 and 65 is 110
The product of 45 and 65 is 2925
The sum of 45 and 65 is 110 and product of 45 and 65 is 2925
```

In Program 10, we accept numbers in hexadecimal, octal and binary systems and display their equivalent decimal numbers.

Program

Program 10: A Python program to convert numbers from other systems into decimal number system.

```
# input from other number systems
str = input('Enter hexadecimal number: ')  # accept input as string
n = int(str, 16)  # inform the number is base 16
print('Hexadecimal to Decimal= ', n);

str = input('Enter octal number: ')
n = int(str, 8)  # inform the number is base 8
print('Octal to Decimal= ', n);

str = input('Enter binary number: ')
n = int(str, 2)  # inform the number is base 2
print('Binary to Decimal= ', n);
```

Output:

```
C:\>python convert.py
Enter hexadecimal number: a
Hexadecimal to Decimal= 10
Enter octal number: 10
Octal to Decimal= 8
Enter binary number: 1101
Binary to Decimal= 13
```

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()]
```

In the previous statement, the input() function will display the message 'Enter two numbers: ' to the user. When the user enters two values, they are accepted as strings. These strings are divided wherever a space is found by split() method. So, we get two strings as elements of a list. These strings are read by for loop and converted into integers by the int() function. These integers are finally stored into a and b.

The spilt() method by default splits the values where a space is found. Hence, while entering the numbers, the user should separate them using a space. The square brackets [ ] around the total expression indicates that the input is accepted as elements

of a list. Program 11 shows how to accept 3 integer numbers from in the same line. While entering the numbers, the user should separate them with at least one space.

Program

Program 11: A Python program to accept 3 integers in the same line and display their sum.

```
# accepting 3 numbers separated by space
var1, var2, var3 = [int(x) for x in input("Enter three numbers: ").split()]
print('Sum = ', var1+var2+var3)
```

Output:

```
C:\>python Input.py
Enter three numbers: 10 20 30
Sum = 60
```

Suppose the user wants to separate the input with commas while entering them, we can specify a comma for the split() method as shown in Program 12.

Program

Program 12: A Python program to accept 3 integers separated by commas and display their sum.

```
# accepting 3 numbers separated by comma.
var1, var2, var3 = [int(x) for x in input("Enter three numbers:").
                    split(',')]
print('Sum = ', var1+var2+var3)
```

Output:

```
C:\>python Input.py
Enter three numbers: 10, 20, 30
Sum = 60
```

Now we will see how to accept a group of strings from the keyboard and display them again. We should remember that the input() function reads given data in the form of strings. Hence the variable 'x' in Program 13 takes the strings one by one from the input() function.

Program

Program 13: A Python program to accept a group of strings separated by commas and display them again.

```
# accepting a group of strings from keyboard.
lst = [x for x in input('Enter strings: ').split(',')]
print('You entered:\n', lst)
```

Output:

```
C:\>python Input.py
Enter strings: Anil, Vijay Kumar, Priya
You entered:
['Anil', 'Vijay Kumar', 'Priya']
```

The eval() function takes a string and evaluates the result of the string by taking it as a Python expression. For example, let's take a string: "a+b-4" where a = 5 and b = 10. If we pass the string to the eval() function, it will evaluate the string and returns the result. Consider the following example:

```
a, b = 5, 10
result = eval("a+b-4")
print(result)
11
```

We can use the eval() function along with input() function. Since the input() function accepts a value in the form of a string, the eval() function receives that string and evaluates it. In Program 14, we can enter an expression that is evaluated by the eval() function and result is displayed.

Program

Program 14: Evaluating an expression entered from keyboard.

```
# using eval() along with input() function
x = eval(input("Enter an expression: "))
print("Result= %d " % x)
```

Output:

```
C:\>python Input.py
Enter an expression: 10 + 5 - 4
Result= 11
```

We can use the combination of eval() and input() functions to accept objects like lists or tuples. When the user types the list using square braces [ ], eval() will understand that it is a list, as shown in Program 15:

Program

Program 15: A Python program to accept a list and display it.

```
# accepting a list from keyboard
lst = eval(input("Enter a list: "))
print("List= ", lst)
```

Output:

```
C:\>python Input.py
Enter a list: ["Ajay", "Preethi", "Sashank", "Vishnu"]
List = ['Ajay', 'Preethi', "Sashank', 'Vishnu']

C:\>python Input.py
Enter a list: [10, 20, 30]
List = [10, 20, 30]
```

Similarly, when the user types a tuple using parentheses ( ), eval() will understand that it is a tuple, as shown in Program 16.

Program

Program 16: A Python program to accept a tuple and display it.

```
# accepting a tuple from keyboard
tpl = eval(input("Enter a tuple: "))
print("Tuple= ", tpl)
```

Output:

```
C:\>python Input.py
Enter a tuple: (10, 20, 30, 40)
Tuple= (10, 20, 30, 40)
```

Command Line Arguments

We can design our programs in such a way that we can pass inputs to the program when we give run command. For example, we write a program by the name 'add.py' that takes two numbers and adds them. We can supply the two numbers as input to the program at the time of running the program at command prompt as:

```
C:\>python add.py  10  22
Sum = 22
```

Here, add.py is our Python program name. While running this program, we are passing two arguments 10 and 22 to the program, which are called *command line arguments*. So, command line arguments are the values passed to a Python program at command prompt (or system prompt). Command line arguments are passed to the program from outside the program. All the arguments should be entered from the keyboard separating them by a space. These arguments are stored by default in the form of strings in a list with the name 'argv' which is available in *sys* module. Since argv is a list that contains all the values passed to the program, argv[0] represents the name of the program, argv[1] represents the first value, argv[2] represents the second value and so on, as shown in Figure 5.2:

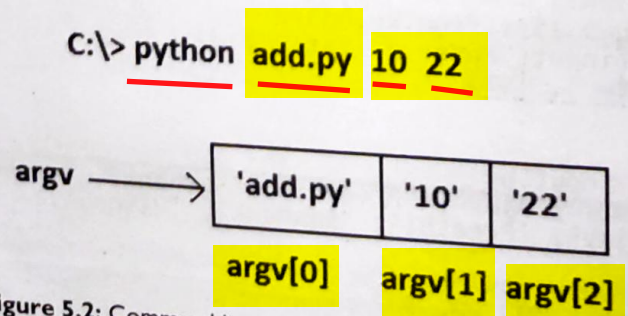

Figure 5.2: Command line args are stored as strings in argv list

If we want to find the number of command line arguments, we can use the len() function as: len(argv). The following program reads the command line arguments entered at command prompt and displays them.

Program 17: A Python program to display command line arguments.

```
# to display command line args. Save this as cmd.py.
import sys

n = len(sys.argv)   # n is the number of arguments
args = sys.argv   # args list contains arguments
print('No. of command line args= ', n)
print('The args are: ', args)
print('The args one by one: ')
for a in args:
    print(a)
```

Output:

```
C:\>python cmd.py 10 Aishwarya Rai 22500.75
No. of command line args=  5
The args are:  ['cmd.py', '10', 'Aishwarya', 'Rai', '22500.75']
The args one by one:
cmd.py
10
Aishwarya
Rai
22500.75
```

Observe the output of the Program 17. Actually, we are passing 3 arguments: 10, Aishwarya Rai and 22500.75. But they are stored as 4 arguments as we can see in the output. The reason is the string 'Aishwarya Rai' has two words and hence it is taken as two arguments instead of one. So, how to make 'Aishwarya Rai' as a single argument? For this purpose, we should enclose this string within quotation marks in either of the two ways shown below:

```
'"Aishwarya Rai"'    # double quotes inside single quotes
"'Aishwarya Rai'"    # single quotes inside double quotes
```

Let's run the Program 17 again and see the output.

```
C:\>python cmd.py 10 "'Aishwarya Rai'" 22500.75
No. of command line args=  4
The args are:  ['cmd.py', '10', "'Aishwarya Rai'", '22500.75']
The args one by one:
cmd.py
10
'Aishwarya Rai'
22500.75
```

In Program 18, we are accepting two numbers from command line and finding their sum. The logic is straight forward. We know that all the command line arguments are stored by default in argv. So, argv[0] represents our program name. argv[1] represents the first number and argv[2] represents the second number entered at command prompt. Since by default, all command line arguments are stored in the form of strings, we can convert them into numeric format using int() or float() functions as:

```
int(sys.argv[1])   # converts argv[1] into int type
float(sys.argv[2])   # converts argv[2] into float type
```

Once the arguments are available as numbers, we can perform any arithmetic operations on them.

Program

Program 18: A Python program to find sum of two numbers using command line arguments.

```
# to add two numbers. Save this as add.py
import sys

# convert args into integers and add them
sum = int(sys.argv[1])+int(sys.argv[2])
print('Sum= ', sum)
```

Output:

```
C:\>python add.py  10 22
Sum= 22
```

We will write another program where we enter some numbers and find the sum of even numbers. This is shown in Program 19.

Program

Program 19: A Python program to find the sum of even numbers using command line arguments.

```
# to find sum of even numbers
import sys

# read command line arguments except the program name
args = sys.argv[1:]
print(args)

sum=0
# find sum of even arguments
for a in args:
    x = int(a)
    if x%2==0:
        sum+=x

print('Sum of evens= ', sum)
```

Output:

```
C:\>python cmd.py 6 8 9 10 11
['6', '8', '9', '10', '11']
Sum of evens= 24
```