Programming Fundamentals (SWE – 102)

Formatted input/output

Why taking user inputs?

- Developers often have a need to interact with users, either to get data or to provide some sort of result.

- Most programs today use a dialog box as a way of asking the user to provide some type of input.

Formatted I/O

- Formatted output converts the internal binary representation of the data to ASCII characters which are written to the output file.
- Formatted input reads characters from the input file and converts them to internal form.
- Formatted input/output is very portable. It is a simple process to move formatted data files to various computers, even computers running different operating systems, if they all use the ASCII character set.
- **print()** and **input()** are examples for formatted input and output functions.

Reading Input From the Keyboard

- Programs often need to obtain data from the user, usually by way of input from the keyboard.
- The simplest way to accomplish this in Python is with **input().**

input([<prompt>])

"Reads a line of input from the keyboard"

- **input()** pauses program execution to allow the user to type in a line of input from the keyboard. Once the user presses the Enter key, all characters typed are read and returned as a string.

Example#1:

- name =input('What is your name?')

“name” is a variable

“input()” is a formatted function

('What is your name?') is a prompt or passing string.

Output:

```
>>> %Run EX1.py
What is your name?XYZ
>>>
```

```
14
15 name =input('What is your name?')
16
17
```

Shell

```
>>> %Run 'input methods.py'
  What is your name?abc
>>>
```

Example#2:how to display input value

```
#Description: Program to check input
#Taking input from user
num = input('Enter a number: ')
#Display input value using Print()
print("Input number:“ ,num)
```

Output:

```
>>> %Run EX2.py
Enter a number: 2
Input number: 2
>>>
```

display input value

```
12
13  #Taking input from user
14  num = input('Enter a number: ')
15
16  #Display input value using Print()
17  print("Input number:" ,num)
18
19
```

Shell

```
>>> %Run 'input methods.py'
  Enter a number: 10
  Input number: 10
>>>
```

Display **Value** in end of the statement which is stored in variable **name**

```
1  print('Enter your name:')
2  name= input()
3  print('Hello,' ,name)
4
```

Shell

```
>>> %Run 'input methods.py'
  Enter your name:
  john
  Hello, john
```

display name = input value in middle

```
4
5  print('Enter your name:')
6  name= input()
7  print('Hello,' ,name, "Have a Good Day")
8
```

Shell

```
>>> %Run 'input methods.py'
  Enter your name:
  john
  Hello, john Have a Good Day

>>>
```

display name = input value in start

```
 8
 9  print('Enter your name:')
10  name= input()
11  print(name,'Hello, Have a Good Day!')
12
```

Shell

```
>>> %Run 'input methods.py'
 Enter your name:
 john
 john Hello, Have a Good Day!

>>>
```

Display name with + Operator

```
 8
 9  print('Enter your name:')
10  name= input()
11  print('Hello, Have a Good Day! ' + name)
12
```

Shell

```
>>> %Run 'input methods.py'
  Enter your name:
  john
  Hello, Have a Good Day! john
>>>
```

Is + operator work in start ?

```
 8
 9  print('Enter your name:')
10  name= input()
11  print(+name 'Hello, Have a Good Day! ')
12
```

Shell

```
>>> %Run 'input methods.py'
  Traceback (most recent call last):
    File "D:\input methods.py", line 11
      print(+name 'Hello, Have a Good Day! ')
                                          ^
  SyntaxError: invalid syntax

>>>
```

Is + operator work in middle?

```
 8
 9  print('Enter your name:')
10  name= input()
11  print('Hello,' +name 'Have a Good Day!')
12
```

Shell

```
>>> %Run 'input methods.py'
  Traceback (most recent call last):
    File "D:\input methods.py", line 11
      print('Hello,' +name 'Have a Good Day!')
                                             ^
  SyntaxError: invalid syntax

>>>
```

Typecasting in Python

- input() always returns a string. If you want a numeric type, then you need to convert the string to the appropriate type with the int(), float(), or complex() built-in functions. This is called typecasting.

- Typecasting is when you convert a variable value from one data type to another.

Input datatype in python By default (String)

```
19
20  num= input ("Enter number :")
21  print(num)
22  name = input ("Enter name : ")
23  print(name)
24
25  # Printing type of input value
26  print ("type of number", type(num))
27  print ("type of name", type(name))
28
```

Shell

```
>>> %Run 'input methods.py'
  Enter number :10
  10
  Enter name : john
  john
  type of number <class 'str'>
  type of name <class 'str'>
>>>
```

Without Typecasting Example 1

```
28
29  num1 = (input("Enter first number: "))
30  print(num1)
31  num2 = (input("Enter Second number: "))
32  print(num2)
33
34  # printing the sum in integer
35  print("The addition of two numbers is: ",num1 + num2)
```

Shell

```
>>> %Run 'input methods.py'
  Enter first number: 3
  3
  Enter Second number: 7
  7
  The addition of two numbers is:  37
>>>
```

Example2 : Typecasting

- n = input('Enter a number: ')
- print(n + 100) # adding number into string

Output:

- Enter a number: 50
- Traceback (most recent call last):
- File "<stdin>", line 1, in <module>
- TypeError: must be str, not int

- **n = int(input('Enter a number: '))**
- print(n + 100)
- Enter a number: 50
- 150

Without Type Casting Example:2

```
38
39 n = input('Enter a number: ')
40 print(n + 100) # adding number into string
41
```

Shell

```
>>> %Run 'input methods.py'
 Enter a number: 7
 Traceback (most recent call last):
   File "D:\input methods.py", line 40, in <module>
     print(n + 100) # adding number into string
 TypeError: can only concatenate str (not "int") to str
```

Type Casting (Integer)

```
42  n = int(input('Enter a number: '))
43  print(n + 100)
44
45
```

Shell

```
>>> %Run 'input methods.py'
  Enter a number: 7
  107

>>>
```

Type Casting integer to float

```
42 n = int(input('Enter a number: '))
43 print(n + 100)
44
45
```

Shell

```
Python 3.7.9 (bundled)
>>>
>>> %Run 'input methods.py'
 Enter a number: 3.5
 Traceback (most recent call last):
   File "D:\input methods.py", line 42, in <module>
     n = int(input('Enter a number: '))
 ValueError: invalid literal for int() with base 10: '3.5'
```

Type Casting (Float)

```
42 n = float(input('Enter a number: '))
43 print(n + 100)
44
45
```

Shell

```
>>> %Run 'input methods.py'
  Enter a number: 3.5
  103.5
```

Taking multiple inputs from user in Python

- Developer often wants a user to enter multiple values or inputs in one line.

- In C++/C user can take multiple inputs in one line using scanf() but in Python user can take multiple values or inputs in one line by the method called split() method.

Split() method

- This function helps in getting a multiple inputs from user. It breaks the given input by the specified separator.

Syntax :

input().split(separator)

```
# Description: Python program showing how to multiple input using split
# Taking two inputs at a time
x, y = input("Enter a two value: ").split()
# Display inputs
print("Number of boys: ", x)
print("Number of girls: ", y)
```

Split() Function

```
44
45 # Description: Python program showing how to multiple input using split
46 # Taking two inputs at a time
47 x, y = input("Enter a two value: ").split()
48 # Display inputs
49 print("Number of boys: ", x)
50 print("Number of girls: ", y)
51
52
53
54
```

Shell

```
>>> %Run 'input methods.py'
 Enter a two value: 7 10
 Number of boys:  7
 Number of girls:  10
```

Displaying Output/Formatted output

- The print() function prints the specified message to the screen.

- The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax:

print(*object(s)*, sep=*separator*, end=*end*)

Print()

print() Parameters:

- objects - object to the printed. * indicates that there may be more than one object
- sep - objects are separated by sep. Default value: ' '
- end - end is printed at last

Example 4: using print()

- Print more than one object:

```
print("Hello", "how are you?")
```

- Three objects are passed:

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

- Print two messages, and specify the separator:

```
print("Hello", "how are you?", sep="---")
```

Object

```
1 name = 'john'
2 print("Hello" ,name, "how are you?","Good Morning")
3
```

Shell

```
>>> %Run objects.py

 Hello john how are you? Good Morning
```

Separator

```
print ("Hello","How are you?")
```

```
Shell
Python 3.7.9 (bundled)
>>> %Run sep.py
 Hello How are you?
```

By default "space"

- Print two messages, and specify the separator:

print("Hello", "how are you?", sep="---")

```
print("Hello" ,"how are you?", sep='---')
```

```
>> %Run test.py
Hello---how are you?
```

Example : using print()

- print() with separator and end parameters:

```
a = 5
print("a =", a, sep='0000', end='\n\n')
print("a =", a, sep='0', end='')
```

Separators()

```
1  print("Hello", "how are you?", sep='@')
2  print("Hello", "how are you?", sep='#')
3  print("Hello", "how are you?", sep='*')
4  print("Hello", "how are you?", sep='%')
5  print("Hello", "how are you?", sep='&')
6  print("Hello", "how are you?", sep='\n')
7  print("Hello", "how are you?", sep='\t')
8  print("Hello", "how are you?", sep='--')
9  print("Hello", "how are you?", sep='_')
10 print("Hello", "how are you?", sep=':')
11
12
```

hell ×

```
>>> %Run sepratators.py
Hello@how are you?
Hello#how are you?
Hello*how are you?
Hello%how are you?
Hello&how are you?
Hello
how are you?
Hello    how are you?
Hello--how are you?
Hello_how are you?
Hello:how are you?
```

End

- Print function has end parameter
- By default end =“ \n” has new line escape sequence

```
2
3  print("Enter your name")
4  print("Enter your Section")
5
```

Shell

```
>>> %Run sep.py
  Enter your name
  Enter your Section
```

Eval()

- The eval() method parses the expression passed to this method and runs python expression (code) within the program.
- Syntax:

```
eval(expression)
```

- Example:

```
x = 1
print(eval('x + 1')) or  print(eval("x + 1"))
```

Output: 2

```
1  x = 1
2  print(eval("x + 1"))
3
```

Shell

```
>>> %Run evaluate.py
 2
>>>
```

EXAMPLE 1

```
1
2  print(eval("5 * 5"))
3
```

Shell

```
>>> %Run evaluate.py
 25
>>>
```

EXAMPLE 2

```
1  a,b,c=4, 3, 5
2  print(eval("(a**2)+ (b/2)- (c*3)"))
3
```

Shell

```
>>> %Run evaluate.py
 2.5
```

EXAMPLE 3

```
1
2  print(eval("(3 + 4) * (4 + 5) * (4 + 3) - 1"))
3
```

hell

```
>>> %Run evaluate.py
 440
```

EXAMPLE 4

EXAMPLE 5

```
1
2 print(eval("(2 % 2) + (2 * 2) - (2 / 2)"))
3
```

Shell

```
>>> %Run evaluate.py
 3.0
>>>
```

Task

- Take three input as integers and displays their average using eval()

Practice Question:

- Write a program that reads a Celsius degree from the console and converts it to Fahrenheit and displays the result. The formula for the conversion is as follows:
- fahrenheit = (9 / 5) * celsius + 32
- Here is a sample run of the program:
- Enter a degree in Celsius:43
- 43 Celsius is 109.4 Fahrenheit