INPUT AND OUTPUT

Read dynamic data from the keyboard

Python-2

- raw_input() #input is always considered as str and typecasting is required
- input() #input is not considered as str and no typecasting is required

Python-3

No raw_input() is available

- input() #input is always considered as str and typecasting is required

```
x = input("Enter any number : ")
print(type(x))
# output
Enter any number : 10
<class 'str'>

# You can typecast the input using typecasting functions
```

Reading Multiple Inputs

- Using split() method
- Using List comprehension

1. split() Method

This function helps in getting multiple inputs from users. It breaks the given input by the specified separator. If a separator is not provided then any white space is a separator. Generally, users use a split() method to split a Python string but one can use it in taking multiple inputs.

Syntax:
input().split(separator, maxsplit)

```
# taking two inputs at a time

x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)

# taking three inputs at a time
x, y, z = input("Enter three values: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
print("Number of girls is : ", z)

# taking two inputs at a time
a, b = input("Enter two values: ").split()
print("First number is {} and second number is {}".format(a, b))

# taking multiple inputs at a time
# and type casting using list() function
x = list(map(int, input("Enter multiple values: ").split()))
print("List of students: ", x)



# output
Enter two values: 5 10
Number of boys:  5
Number of girls:  10
Enter three values: 5 10 15
Total number of students:  5
Number of boys is :  10
Number of girls is :  15
Enter two values: 5 10
First number is 5 and second number is 10
Enter multiple values: 5 10 15 20 25
List of students:  [5, 10, 15, 20, 25]
```

--

2. Using List comprehension

List comprehension is an elegant way to define and create a list in Python. We can create lists just like mathematical statements in one line only. It is also used in getting multiple inputs from a user.

```
# taking two input at a time
x, y = [int(x) for x in input("Enter two values: ").split()]
print("First Number is: ", x)
print("Second Number is: ", y)

# taking three input at a time
x, y, z = [int(x) for x in input("Enter three values: ").split()]
print("First Number is: ", x)
print("Second Number is: ", y)
print("Third Number is: ", z)

# taking two inputs at a time
x, y = [int(x) for x in input("Enter two values: ").split()]
```

```
print("First number is {} and second number is {}".format(x, y))

# taking multiple inputs at a time
x = [int(x) for x in input("Enter multiple values: ").split()]
print("Number of list is: ", x)

# output

Enter two values: 5 10
First Number is:  5
Second Number is:  10
Enter three values: 5 10 15
First Number is:  5
Second Number is:  10
Third Number is:  15
Enter two values: 5 10
First number is 5 and second number is 10
Enter multiple values: 5 10 15 20 25
Number of list is:  [5, 10, 15, 20, 25]
```

Note:

The above examples take input separated by spaces. In case we wish to take input separated by comma
(,), we can use the following:

```
# taking multiple inputs at a time separated by comma
x = [int(x) for x in input("Enter multiple value: ").split(",")]
print("Number of list is: ", x)
```

--

eval() function

Python eval() function parse the expression argument and evaluate it as a python expression and runs python expression (code) within the program.

Syntax:
eval(expression, globals=None, locals=None)

Parameters:
expression: String is parsed and evaluated as a Python expression globals [optional]: Dictionary to specify the available global methods and variables. locals [optional]: Another dictionary to specify the available local methods and variables.

Return:

Returns output of the expression.

```
expression = 'x*(x+1)*(x+2)'
print(expression)

x = 3

result = eval(expression)
print(result)

# Output:

x*(x+1)*(x+2)
60
```

Taking Sequence Using eval()

using input() method:

```
x = input("enter list") #input : [10,20,30]
type(x) #output : <class 'str'>
```

using eval() method:

```
x = eval(input("ENTER DATA")) #input : 10
type(x)                       #output: int



x = eval(input("ENTER DATA")) #input : 10.1
type(x)                       #output: float



x = eval(input("ENTER DATA")) #input : (12,34,5)
type(x)                       #output: tuple



x = eval(input("ENTER DATA")) #input : [10,20,30]
type(x)                       #output: list

a,b,c = [eval(x) for x in input("Enter Three Values").split()]
list1 = [eval(x) for x in input("Enter Three Values").split(",")]
```

Command Line Arguments

The arguments that are given after the name of the program in the command line shell of the operating system are known as Command Line Arguments. Python provides various ways of dealing with these types of arguments. The three most common are:

- Using sys.argv
- Using getopt module
- Using argparse module

Ye wali link se padh le

Output Statements

Python print() function prints the message to the screen or any other standard output device.

```
Form-1 : print() #No arguments ;prints new line
Form-2 : print("string argument")
Form-3 : print("str1"+"str2") #concatenates str1 and str2 ;both args should be
         string type
Form-4 : print("str" * n) #string repeation ;prints string n times
Form-5 : print(variableNumberOfArgs) #print("hello",2,3)

Form-6 : using sep attribute
*sep attribute*
print(12,34,5)          #output : 12 34 5
print(12,34,5,sep=',')  #output : 12,34,5

Form-7 : end attribute
*end attribute*
print(10)
print(30)
print(4)

# output
10
30
4

print(10,end="")
print(20,end="")
print(30,end="")
```

```
# output
10 20 30
```

Form-8 : Formatted string

In Python, there are several ways to present the output of a program. Data can be printed in a human-readable form, or written to a file for future use, or even in some other specified form. Users often want more control over the formatting of output than simply printing space-separated values.

Output Formatting in Python There are several ways to format output using String Method in Python.

- Using String Modulo Operator(%)
- Using Format Method
- Using The String Method
- Python’s Format Conversion Rule

Yha se padh le