2. Input and Output

1. Reading dynamic input from the keyboard

Python 2

In Python 2 the following 2 functions are available to read dynamic input from the keyboard.

1. raw_input()
2. input()

1. raw_input():

This function always reads the data from the keyboard in the form of String Format. We have to convert that string type to our required type by using the corresponding type casting methods.

2. input():

input() function can be used to read data directly in our required format.We are not required to perform type casting.

Python 3

in Python 3 we have only input() method and raw_input() method is not available.

input():

Python3 input() function behaviour exactly same as raw_input() method of Python2.
i.e
every input value is treated as str type only.

raw_input() function of Python 2 is renamed as input() function in Python3

In [1]:
```
help(input)
```

```
Help on method raw_input in module ipykernel.kernelbase:

raw_input(prompt='') method of ipykernel.ipkernel.IPythonKernel instance
    Forward raw_input to frontends

    Raises
    ------
    StdinNotImplementedError if active frontend doesn't support stdin.
```

In [2]:
```
type(input("Enter Value"))
```

```
Enter Value10
```

Out[2]:
```
str
```

In [3]:
```
type(input("Enter Value"))
```

```
Enter Value10.5
```

Out[3]:
```
str
```

In [4]:
```
type(input("Enter Value"))
```

```
Enter ValueTrue
```

Out[4]:
```
str
```

Q. Write a program to read 2 numbers from the keyboard and print sum.

In [6]:
```
x = input("x: ")     # "9"
y = input("y: ")     # "12"
r = x + y
print("Result = ",r)
```

```
x: 9
y: 12
Result =  912
```

In [5]:
```
x = int(input("x: ")) # 9
y = int(input("y: ")) # 12
print(f"{x} + {y} = {x+y}")
```

```
x: 9
y: 12
9 + 12 = 21
```

In [8]:
```
print("The Sum:",int(input("Enter First Number:"))\
      +int(input("Enter Second Number:")))
```

```
Enter First Number:9
Enter Second Number:12
The Sum: 21
```

In [20]:
```
name = input("name: ")
birthyear = int(input("birthyear: "))
s = f"Hello user {name}, you are {2021 - birthyear} years old"
print(s)
```

```
name: Pankaj
birthyear: 2002
Hello user Pankaj, you are 19 years old
```

How to read multiple values from the keyboard in a single line

In [12]:
```
a,b = [int(x) for x in input("Enter two numbers :").split()]
print("Product is : ",a*b)
```

```
Enter two numbers :12 6
Product is :  72
```

In [13]:
```
a,b = map(int, input("Enter two numbers :").split())
print("Product is : ",a*b)
```

```
Enter two numbers :12 3
Product is :  36
```

In [14]:

```
l1 = list(map(int, input().split()))
print(l1)
```

```
1 2 3 4 5
[1, 2, 3, 4, 5]
```

Q. Write a program to read 3 float numbers from the keyboard with , seperator and print their sum.

In [16]:
```
a,b,c = [float(x) for x in input("Enter 3 float numbers : ").split(',')]
print("The Sum is:",a+b+c)
```

```
Enter 3 float numbers : 12.5,10.6,4.7
The Sum is: 27.8
```

In [ ]:
```

```

eval():

eval Function take a String and evaluate the Result

In [18]:
```
x = eval("10+20+30")
print(x)
```

```
60
```

In [20]:
```
x = eval(input("Enter Expression : "))
print(x)
```

```
Enter Expression : 10+2*3/4
11.5
```

eval() can evaluate the Input to list, tuple, set, etc based the provided Input.

In [23]:
```
l = eval(input("ENter list"))
print(l,type(l))
```

```
ENter list[1, 2, 3, 4]
[1, 2, 3, 4] <class 'list'>
```

2. Command Line Argument

argv is not Array it is a List. it is available in sys Module

the argument which are Passing at Time of execution are called Command Line Arguments

Eg - !python test.py 10 20 30
                    | | |
            Command Line Arguments

Within the Python Program this Command Line Argument are available in argv. Which is present in SYS Module

test.py	10	20	30

Note: argv[0] represents Name of Program. But not first Command Line Argument.
argv[1] represent First Command Line Argument.

In [26]:
```
from sys import argv
print(type(argv))
```

```
<class 'list'>
```

Write a Program to display Command Line Arguments

In [43]:
```
%%writefile test.py
from sys import argv
print("The Number of Command Line Arguments:", len(argv))
print("The List of Command Line Arguments:", argv)
print("Command Line Arguments one by one:")
for x in argv:
    print(x)
```

```
Overwriting test.py
```

In [46]:
```
!python test.py 10 20 30
```

```
The Number of Command Line Arguments: 4
The List of Command Line Arguments: ['test.py', '10', '20', '30']
Command Line Arguments one by one:
test.py
10
20
30
```

Note1: Within the Python program command line arguments are available in the String form. Based on our requirement,we can convert into corresponding type by using type casting methods

In [62]:
```
%%writefile test.py
from sys import argv
sum = 0
args = argv[1:]
for x in args:
    sum = sum + int(x)
print("The Sum : ",sum)
```

```
Overwriting test.py
```

In [63]:
```
!python test.py 10 20 30
```

```
The Sum :  60
```

Note2: usually space is seperator between command line arguments. If our command line argument itself contains space then we should enclose within double quotes(but not single quotes)

In [64]:
```
%%writefile test.py
from sys import argv
print(argv[1])
```

```
Overwriting test.py
```

In [65]:
```
!python test.py Pankaj Kumar
```

```
Pankaj
```

In [66]:

```
!python test.py 'Pankaj Kumar'
```

```
'Pankaj
```

In [67]:
```
!python test.py "Pankaj Kumar"
```

```
Pankaj Kumar
```

Note3: If we are trying to access command line arguments with out of range index then we will get Error.

In [74]:
```
%%writefile test.py
from sys import argv
print(argv[2])
print(argv[100])
```

```
Overwriting test.py
```

In [75]:
```
!python test.py 10 20 30
```

```
20
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    print(argv[100])
IndexError: list index out of range
```

Note: In Python there is argparse module to parse command line arguments and display some help messages whenever end user enters wrong input.

3. Output Statements

We can use print() function to display output.

In [89]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Form-1: print() without any argument

Just it prints new line character

Form-2:

print(String)

In [77]:
```
print("Hello World")
```

```
Hello World
```

We can use escape character alse

In [78]:
```
print("Hello \n World")
```

```
Hello
 World
```

In [79]:
```
print("Hello \t World")
```

```
Hello    World
```

We can use repetetion operator (*) in the string

In [81]:
```
print(5*"Hello ")
```

```
Hello Hello Hello Hello Hello
```

We can Use + operator

In [86]:
```
print("Pankaj"+"Kumar")
```

```
PankajKumar
```

In [85]:
```
print("Pankaj","Kumar")
```

```
Pankaj Kumar
```

Note:

If both arguments are String type then + operator acts as concatenation operator.

If one argument is string type and second is any other type like int then we will get Error

If both arguments are number type then + operator acts as arithmetic addition operator

In [84]:
```
print(10+20)
```

```
30
```

In [87]:
```
print("Pankaj"+10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10516/2292995960.py in <module>
----> 1 print("Pankaj"+10)

TypeError: can only concatenate str (not "int") to str
```

Form-3: print() with variable number of arguments:

In [88]:
```
a,b,c = 10,20,30
```

```
print("The Value are : ",a,b,c)
```

```
The Value are :  10 20 30
```

By default output values are seperated by space.If we want we can specify seperator by using "sep" attribute

In [91]:
```
a,b,c=10,20,30

print(a,b,c,sep=',')
print(a,b,c,sep=':')
```

```
10,20,30
10:20:30
```

Form-4: print() with end attribute:

In [92]:
```
print("Hello")
print("Pankaj")
print("Yadav")
```

```
Hello
Pankaj
Yadav
```

If we want output in the same line with space

In [93]:
```
print("Hello",end='')
print("Pankaj",end='')
print("Yadav")
```

```
HelloPankajYadav
```

Note: The default value for end attribute is \n,which is nothing but new line character

Form-5: print(object) statement:

We can pass any object (like list,tuple,set etc)as argument to the print() statement

In [95]:
```
l=[10,20,30,40,50]
t=(10,20,30,40)
print(l)
print(t)
```

```
[10, 20, 30, 40, 50]
(10, 20, 30, 40)
```

Form-6: print(String,variable list):

We can use print() statement with String and any number of arguments.

In [96]:
```
s = "Pankaj"
a = 20
s1 = "Python"
s2 = "C++"
print("Hello",s,"Your Age is",a)
print("You are learning",s1,"and",s2)
```

```
Hello Pankaj Your Age is 20
You are learning Python and C++
```

Form-7: print(formatted string)

```
%i ====>int
%d ====>int
%f ====>float
%s ====>String type
```

Syntax:

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

In [97]:

```
a = 10
b = 20
c = 30

print("a is %i"%a)
print("b is %d and c is %d"%(b,c))
```

```
a is 10
b is 20 and c is 30
```

In [98]:

```
s = "Pankaj"
l = [10,20,30,40]
print("Hello %s .. The List of items are % s"%(s,l))
```

```
Hello Pankaj .. The List of items are [10, 20, 30, 40]
```

Form-8: print() with replacement operator {}

In [109…

```
name = "Pankaj"
salary=0
work = "Student"

print("Hello {0} your salary is {1} and u are a {2}".\
      format(name,salary,work))

print("Hello {x} your salary is {y} and u are a {z}"\
      .format(x=name,y=salary,z=work))
```

```
Hello Pankaj your salary is 0 and u are a Student
Hello Pankaj your salary is 0 and u are a Student
```

In [ ]: