Python I/O

Python input:

Python, by default, takes the input as a string datatype.

Example 1:

```
x = 10
y= input('Enter a number: ')
print(type(x))
print(type(y))
```

Output:

```
Enter a number: 10
<class 'int'>
<class 'str'>
```

Example 2:

```
a=input('Enter first number: ')
b= input('Enter second number: ')
print(a+b)
x =int(input('Enter first number: '))
y= int(input('Enter second number: '))
print('The sum is :',  x+y )
```

Output:

```
Enter first number: 10
Enter second number: 20
1020
Enter first number: 10
Enter second number: 20
The sum is : 30
```

Example 2's way is considered a good/recommended practice to type cast.

Example 3:

```
roll = int(input ("Enter student roll number : "))
marks = float(input ("Enter student marks : "))
name = input("Enter student name : ")
isPass = bool(input ("Enter True for pass and False for fail : "))
print("These are the student information: ")
```

```
print("Student roll number: ", roll)
print("Student total marks: ",marks)
print("Student name : ", name)
print("Student result: ", isPass)
```

Output:

```
Enter student roll number : 2
Enter student marks : 40
Enter student name : Ram
Enter True for pass and False for fail : False
These are the student information:
Student roll number:  2
Student total marks:  40.0
Student name :  Ram
Student result:  True
```

In the Example 3 above, we see that even if we've entered False, the result shows True. This is because in type-conversion incase of bool, if we input 0 or an empty string, only then will it show False, otherwise every other input will be labeled True.

So for the solution to this, we can use the function eval as:

```
roll = int(input ("Enter student roll number : "))
marks = float(input ("Enter student marks : "))
name = input("Enter student name : ")
isPass = eval(input ("Enter True for pass and False for fail : "))
print("These are the student information: ")
print("Student roll number: ", roll)
print ("Student total marks: ",marks)
print("Student name : ", name)
print("Student result: ", isPass)
```

Output:

```
Enter student roll number : 1
Enter student marks : 300.4
Enter student name : Ramesh
Enter True for pass and False for fail : False
These are the student information:
Student roll number:  1
Student total marks:  300.4
Student name :  Ramesh
Student result:  False
```

A recommended practice to input multiple values together:

```
a = input('Enter two numbers: ')
print('a = ',a)
print(type(a))
b = a.split()
print('b = ', b)
print(type(b))
```

Output:

```
Enter two numbers: 1 2
a =  1 2
<class 'str'>
b =  ['1', '2']
<class 'list'>
```

Here, the string is stored in a list by splitting.

Now, to convert each string element in the list to int:

```
a = input('Enter two numbers: ')
print('a = ',a)
print(type(a))
b = a.split()
print('b = ', b)
print(type(b))
c= [int (x) for x in b]
print('c =',c)
print(type(c))
x, y = c
print("The sum is: ", x+y)
```

Output:

```
Enter two numbers: 1 2
a =  1 2
<class 'str'>
b =  ['1', '2']
<class 'list'>
c = [1, 2]
<class 'list'>
The sum is: 3
```

Here, even though both ‘b’ and ‘c’ are lists, one stores string elements while the other stores integers and hence, we can carry out arithmetic operations.
The above example consists of concepts of list comprehension and unpacking of list.

Another way to go about the above example:

```
a,b= [int(x) for x in input('Enter two numbers: ').split()]
print('Sum is:',a+b)
```

Output:

```
Enter two numbers: 1 2
Sum is: 3
```

- **eval() function:**

  The function evaluates the type of data in input and gives the converted output from string accordingly.

  For example:

```
a = eval(input("Enter anything: "))
print(a,type(a))
```

  Output:

```
Enter anything: 10
10 <class 'int'>
Enter anything: 10.4
10.4 <class 'float'>
Enter anything: True
True <class 'bool'>
Enter anything: [1,2,3]
[1, 2, 3] <class 'list'>
```

  Example2:

```
a = eval('1+2+3')
print(a,type(a))
```

  Output:

```
6 <class 'int'>
```

- **Command Line Argument:**

  In terminal, if along with commands, we give some value/argument then it is called command line argument.

  For example:

```
from sys import argv
print(type(argv))
```

  Output:

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

  If we run the python as:
  python session7.py 10 20

argv will make a list but the list's first value will not be 10 but the file name, such that, argv[0] will give session7.py, argv[1] will give 10 and argv[2] will give 20.

Example:

```
from sys import argv
print(type(argv))
print(argv)
print(argv[1:])
print('The number of command line argument', len(argv))
print('The command line arguments are ',argv)
print('The command line arguments one by one ')
for x in argv:
      print(x)
```

Output:

```
<class 'list'>
['/Users/ishhh/Downloads/Intern/Practice_python/session7.py',10,
20,30]
[10, 20, 30]
The number of command line argument 4
The command line arguments are
['/Users/ishhh/Downloads/Intern/Practice_python/session7.py', 10, 20,
30]
The command line arguments one by one
/Users/ishhh/Downloads/Intern/Practice_python/session7.py
10
20
30
```

Python Output:

The print statement always gives output in a new line with each new print statement. Escape sequences can be used within the print statement itself as well.

Example 1:

```
a=10
print(a)
print()
print('World')
print("hello" + "world")
```

Output:

```
10

World
```

```
helloworld
hello world
hellohellohello
```

- ‘print’ statement can give multiple values at a time as well.

Example 2:

```
a,b,c= 10,20,30
print("The values are ",a,b,c)
print("The values are ",a,b,c, sep=':') //sep separates each value
with the given element
```

Output:

```
The values are 10 20 30
The values are :10:20:30
```

Example 3:

```
print("Python", end=' ')
print("is", end=' ')
print("easy", end='')
```

Output:

```
Python is easy
```

Example 4:

```
name='Ram'
age = 22
print("My name is", name, "and my age is", age)
print("My name is {} and my age is {}".format(name,age))
//indexing the tuple
print("My name is {0} and my age is {1}".format(name,age))
print("My name is {x} and my age is {y}".format(x=name,y=age))
// the best practice to use
print(f"My name is {name} and my age is {age}")
```

Output:

```
My name is Ram and my age is 22
My name is Ram and my age is 22
My name is Ram and my age is 22
My name is Ram and my age is 22
My name is Ram and my age is 22
```

Example 5: (Not generally preferred in python)

```
a=10
b=20
c=30
d= [10,20,30]
e="Ram"
print("Value of a is %d" %a)
print("Value of a is %d and the value of c is %d" %(b,c))
```

Output:

```
Value of a is 10
Value of a is 20 and the value of c is 30
Hello Ram. The list of items are [10, 20, 30]
```