Python Fundamentals day 10

Today's Agenda

- Taking input from command line
- print() function
- Flow of control
- Logical operators
- Comparisons

Taking input from command line

Inputs are also called as arguments. Arguments passed to a program via command line, even before the program begins execution is called as **command line arguments**.

These command line arguments are stored in list, and the name of this list is argv. Anything that is typed after python followed by space is considered as input. Which means even the python file name is also considered as input. And as we know list is ordered, which means the first elements with 0th index is always the name of python file.

But if you try to directly access the elements inside the argv list. Compiler throws error, that's because this list is present inside a module called as sys. So without importing that module you cannot access the elements inside the list.

```
test.py
print(argv[0])
print(argv[1])
print(argv[2])

res = argv[1]/argv[2]
print(res)
```


```
Output Window

C:\python>python test.py 10 2
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    print(argv[0])
NameError: name 'argv' is not defined

C:\python>
```

Let us import sys and see the changes in output

```
test.py
import sys

print(sys.argv[0])
print(sys.argv[1])
print(sys.argv[2])

res = sys.argv[1]/sys.argv[2]
print(res)
Output Window

C:\python>python test.py 10 2
test.py
10
2
Traceback (most recent call last):
  File "test.py", line 7, in <module>
    res = sys.argv[1]/sys.argv[2]
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

The inputs taken through command line are string by default. So we have to change its datatype by type casting before perform any arithmetic operations.

```
test.py
import sys

print(sys.argv[0])
print(sys.argv[1])
print(sys.argv[2])

res = int(sys.argv[1])/int(sys.argv[2])
print(res)
Output Window

C:\python>python test.py 10 2
test.py
10
2
5.0

C:\python>
```

print( ) function

print() is the function that is used the most while coding. Let us see what is the syntax of it and what are the different arguments a print() expects.

Syntax: print(value(s), sep=' ', end='\n', file=file, flush=flush)

Note: Right now let us not concentrate much on the last two arguments file and flush.

Value(s): Accepts multiple input values.

Sep: Separates the values. Separated by spaces(default) if not mentioned any.

End: Ends the line by \n (new line), if not mentioned otherwise.

Let us see some examples

```
In [1]: x=10

In [2]: y=20

In [3]: z=30

In [4]: print(x,y,z)
10 20 30
```

x,y,z are the values here. Separated by spaces and ended by newline. Which are the default values accepted by the arguments.

Let us try changing the default values for separation of values and ending of the line in the following example

```
In [5]: print(x,y,z,sep='*',end='??')
10*20*30??

In [6]:
```

We can now see that the values are separated by * and line has ended by ??.

Note: As this is interactive mode we have been directed to new line. But in case of script mode the cursor would be just after ?? and not in next line.

```
x=10
y=20
z=30
print(x,y,z)
print(x,y,z,sep='*',end='??')
print("enter something")
```

Output:

```
In [8]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
10 20 30
10*20*30??enter something
```

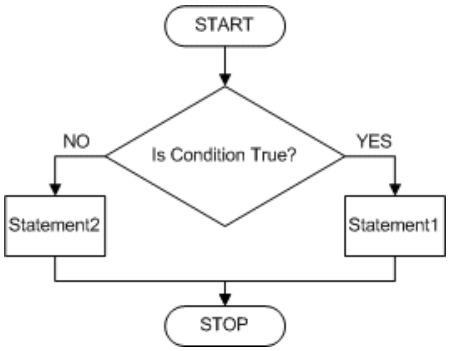
We can see that in script mode second print statement does not end with newline, instead it ends with ?? so as the cursor is after the ?? the next statements gets printed there itself.

Flow of control

The flow of python interpreter is usually sequential. Where it starts from first and executes each line one by one and then ends execution. But there might be certain instances where a set of lines need to be executed only if a certain condition is true/false. As these statements control the flow of program these are called control statements/conditional statements.

We have different conditional statements like:

- if statement
- if else statement
- if elif else statement



if statement

A condition is checked, if it is true the statements under if gets executed, if it is false then it comes out of the conditional statement.

```
if True:
    print("condition is true")
```

Output:

```
In [9]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
condition is true
```

Let us check what happens if the condition is false

```
if False:
    print("condition is true")
```

Output:

```
In [10]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

In [11]:
```


In the above output we can see that nothing gets printed, that's because there is no statement under false condition/else statements which we shall see next.

if else statement

A condition is passed if it is true the statements under if gets executed, if it is false then the statements under else or false condition gets executed.

Let us see an example of checking if a number is even or odd

```
n=int(input("Enter a number \n"))
if n%2==0:
    print(n,"is even")
else:
    print(n,"is odd")
```

Output:

```
In [11]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
12
12 is even

In [12]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
45
45 is odd
```

if elif else statement

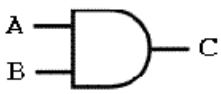
When there are multiple conditions to be checked we use if elif else statement. Elif is a short form of else if.

Before knowing this conditional statement with an example let us see different comparisons that can be done and some logical operators

Logical operators

- and
- or
- not

AND



Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

OR

Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

NOT

Input	Output
A	C
0	1
1	0

Comparisons

Object identity is checks the equality of references of objects.

Equal == checks equality

Less than or equal to <= checks a number is lesser than or equal to the other number

Not equal != checks inequality

Greater than > checks which number is greater

Greater than or equal to >= checks a number is greater than or equal to the other number

Less than < checks which number is lesser

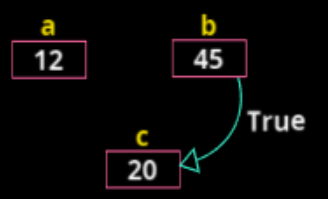
We shall come across many examples where we will be using logical operators and the different comparisons. Let us see an example of if elif else statement

test.py

```
a = int(input("Enter 1st number\n"))
b = int(input("Enter 2nd number\n"))
c = int(input("Enter 3rd number\n"))

if a>b and a>c:
    print(a,"is the largest")
elif b>c:
    print(b,"is the largest")
else:
    print(c,"is the largest")
```

Output Window

```
Enter 1st number
12

Enter 2nd number
45

Enter 3rd number
20
45 is the largest
```



Truth Table

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

In the above example we have seen and operator as well as if elif else statement. Now let's see some other logical operators

OR operator

test.py

```
a = 13

if a%3==0 or a%5==0:
    print(a,"is divisible by either 3 or 5")
else:
    print(a,"is not divisible by neither 3 or 5")
```

Truth Table

a	b	a or b
0	0	0
0	1	1
1	0	1
1	1	1

Output Window

```
In [2]: runfile('C:/python/test.py', wdir='C:/python')
10 is divisible by either 3 or 5

In [3]: runfile('C:/python/test.py', wdir='C:/python')
13 is not divisible by neither 3 or 5

In [4]:
```

NOT operator

test.py

```
n = int(input("Enter a number\n"))

if not (n%2==0):
    print(n,"Odd number")
else:
    print(n,"Even number")
```

Truth Table

a	not a
0	1
1	0

Output Window

```
Enter a number
12
12 Even number

In [3]: runfile('C:/python/test.py', wdir='C:/python')

Enter a number
45
45 Odd number
```

In python we have certain values when passed in conditional statements, they evaluate to be false. These values are called false values which are: False, None, 0, empty sequence (", (), []), empty mapping i.e. {}