Python 101

STD IO and Flow Control (Conditionals).

Contents

Understanding the data streams

Stream means a flow of something.

Data stream means a stream that has data in it. In computers data is in 1/0. So essentially a Data stream is a flow of a lot of 1s and 0s.

Whenever data goes from any one location in OS to some other location it goes through a data stream. Whenever we open a file somewhere in the system it opens into a stream as on opening file the data is transferred from file to the program that has opened the file.

Any stream can be redirected to any direction that we want for it to reach any destination we want.

Understanding the data streams contd..

In an Operating System we have many data streams. Various destinations in Operating Systems are often from various processes that exist in OS.

Out of all the streams that exist in an OS there are three out of them which are of most significance. Those are used by the computer programs to talk to their Environment.

Here environment means the “settings” in which the program is executing. Here Settings refer to a program specific interface provided by OS for interaction between OS and program.

Standard Streams

There are 3 Standard Streams which talk directly with the OS. That are called STDIN, STDOUT and STDERR. In *NIX systems these are streams located at /dev in form of a file.

- STDIN is provided to the program to tell something to the OS.
- STDOUT is provided to the program to hear something from the OS.
- STDERR is provided to the program to acknowledge OS about faults that occurred in program.

Something to ponder on is that whatever we put on STDOUT is seen on terminal and whatever someone writes on terminal can be seen on STDIN.

STD I/O in Python practical

How can we see STDIN, STDOUT and STDERR in python?

These can be accessed using following code-

```
import sys

sys.stdout

sys.stdin

sys.stderr
```

How to write or read from a data stream?

Using **print** we can write to a data stream and Using **input** we can read from a data stream.

The print function practical

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

Whenever we get a paper printed the machine puts ink on the paper. Similarly whenever we make our program print something on a data stream the program puts some 0s and 1s ie. data on that data stream.

*object means print can print any number of objects we pass to it (ie. anything we pass to it as everything is object in python) separated by the “sep” ending with the “end” to a “file” with flush True or False.

Try out the following code on the interpreter

```
>>> print(“hello python”)
hello python ← mind that here exists a new line here.
>>> print(“hello python”, end=”)
hello python>>> ← no new line here because we changed the end to ‘‘ empty
>>> import this
>>> print(this)
<module 'this' from '/usr/lib/python3.6/this.py'> ← it printed this object as well.
>>> print(this, “hello python”)
<module 'this' from '/usr/lib/python3.5/this.py'> hello python ← it printed our two objects separated by a space.
>>> print(this, “hello python”, sep=’ xyz ’)
<module 'this' from '/usr/lib/python3.5/this.py'> xyz hello python ← it printed our two objects separated by ‘ xyz ’
now.
```

What about the file and flush ? We will study about them more in file handling.

The input function practical

```
input([prompt])
```

input means taking something from the user. it takes that “something” from the user from STDIN ie. from terminal.

Here prompt means that what you want to show to the user when you are asking for the input.

eg. input(“Please enter your name\n”)

The input function always returns a string. We will study what are strings in Data Structures in Python (Part - 1).

Flow Control in Python

Flow means how something moves. In Programming languages the Flow means how the execution of code moves. ie. In which sequence the statements we write in a program are executed.

Try this small program

```
print(“Hello”)

print(“world”)
```

We get Hello then World which means if we dont do anything then python is executing our statements as we have written them ie. linearly.

Flow Control in Python contd..

Flow control means Controlling the sequence of execution of those statements. We just observed that if we don’t do anything the execution sequence is linear.

The flow is majorly controlled using two things-

1. conditionals (if, else , elif)
2. loops (for, while)

Conditionals in Python (The if)

There are three conditionals in Python **if, else and elif.**

```
if BOOLEAN_EXPRESSION:

    STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION IS TRUE
```

Boolean expressions are those expression which result in a boolean value (True or False). We already know that expressions that contain **Logical or Relational operators** result in a Boolean.

Now why colon (:) and some space?

A block of code in Python

Just like in C and Java we represent a code block using parenthesis. In Python we represent a block of code using indentation. Indentation means “some space which can be given using tabs or space character”. All the statements under one block of code must have same indentation.

PEP8 says that indentation should be done using 4 spaces and not using tab.

PEP8 also says that we should never mix spaces and tabs.

But why the colon? The colon defines the end of a header and start of a new block of code.

Try out the following code on the interpreter

```
>>> if True:
. . . .    print(“The boolean expression was true so I got printed”)
>>> The boolean expression was true so I got printed
>>> if False:
. . . .    print(“The boolean expression was false so I you will see nothing after pressing enter as I did not get
executed”)
>>> if 1==1:
. . . .    print(“Because 1==1 is True so I will be printed”)
Because 1==1 is True so I will be printed
>>> if 1==2:
. . . .    print(“Because 1==2 is False so I will not be printed”)
>>> x=2
>>> if x==2:
. . . .    print(“Because we just assigned x the value 2 so x==2 is True here so I got printed!”)
Because we just assigned x the value 2 so x==2 is True here so I got printed!
>>> x =1
>>> if x==2:
. . . .    print(“x is not 2 anymore so x==2 is false and I will not be printed”)
```

Conditionals in Python (The else)

```
if BOOLEAN_EXPRESSION:

    STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION IS TRUE

else:

    STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION IS FALSE
```

Try out the following code on the interpreter

```
>>> if False:
. . . .    print(“Boolean Expression was False so I will not be printed”)
. . . .else:
. . . .    print(“But I am inside else so as Boolean Expression is False I will get  printed”)
But I am inside else so as Boolean Expression is False I will get  printed
>>> if 1==1:
. . . .    print(“Because 1==1 is True so I will be printed”)
. . . .else:
           print(“Because 1==1 is True so I will not get printed”)
Because 1==1 is True so I will be printed
```

Conditionals in Python (The else if and elif and Conditional Chaining)

```
if BOOLEAN_EXPRESSION1:

    STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION1 IS TRUE

elif BOOLEAN_EXPRESSION2:

... STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION2 IS TRUE
AND BOOLEAN_EXPRESSION1 WAS FALSE

else:

... STATEMENTS TO EXECUTE IF ALL BOOLEAN_EXPRESSIONS ARE
FALSE
```

Try out the following code on the interpreter

```
>>> if 1<10:
. . . .   print(“Boolean_Expression1is True So only I will be printed”)
. . . .elif 2<10 :
. . . .   print(“Boolean_Expression2 is also True but I will not be printed as Boolean_Expression1 is True”)
. . . . else:
. . . .   print(“I will not be printed as at least one Boolean Expression is True”)
Boolean_Expression1is True So only I will be printed
>>> if 1>10:
. . . .   print(“Boolean_Expression1is False So I will not be printed”)
. . . .elif 2<10 :
. . . .   print(“Boolean_Expression2 is also True and Boolean_Expression1 is False So I will be printed!”)
. . . . else:
. . . .   print(“I will not be printed as at least one Boolean Expression is True”)
Boolean_Expression2 is also True and Boolean_Expression1 is False So I will be printed!
>>> if 1>10:
. . . .   print(“Boolean_Expression1is False So I will not be printed”)
. . . .elif 2>10 :
. . . .   print(“Boolean_Expression2 is also False and So I will also not be printed!”)
. . . . else:
. . . .   print(“Now I will print Because both of you are wrong :-P ”)
. . . . Now I will print Because both of you are wrong :-P
```

A word about the conditional chaining

If we use conditionals in this form

```
if BOOLEAN_EXPRESSION1:
     CODE_BLOCK1
elif BOOLEAN_EXPRESSION2:
     CODE_BLOCK2
elif BOOLEAN_EXPRESSION3:
     CODE_BLOCK3
else:
     CODE_BLOCK4
```

Then always out of CODE_BLOCK1 ,2 ,3 ,4 ...n only 1 Code block will get executed..

A word about the conditional chaining

If we need to execute more than one code blocks then we must set up more than one ifs.

```
if BOOLEAN_EXPRESSION1:
    CODE_BLOCK1
if BOOLEAN_EXPRESSION2:
    CODE_BLOCK2
elif BOOLEAN_EXPRESSION3:
    CODE_BLOCK3
else:
    CODE_BLOCK4
```

Now here CODE_BLOCK1 and 2 both will execute. Because CODE_BLOCK2 only requires BOOLEAN_EXPRESSION2 to be True it doesn't matter if the previous expressions were true or false.

Try out the following code on the interpreter

```
>>> if 1<10:
. . . .    print(“Boolean_Expression1is True So I will be printed”)
. . . .if 2<10 :
. . . .    print(“Boolean_Expression2 is also True so I will also be printed irrespective of whether 1 is true or false”)
. . . .elif 3<10:
. . . .    print(“Boolean_Expression3 is also True but will not be printed because its corresponding if is already true”)
. . . .else:
. . . .   print(“No one gives a damn about me”)
Boolean_Expression1is True So only I will be printed
Boolean_Expression2 is also True so I will also be printed irrespective of whether 1 is true or false
```

Nesting of Conditionals

Writing the Conditional Chain inside an existing Conditional Chain is called Nesting of Conditionals.

```
if BOOLEAN_EXPRESSION1:
    if BOOLEAN_EXPRESSION11:
        CODE_BLOCK11
    else:
        CODE_BLOCK12
elif BOOLEAN_EXPRESSION2:
    CODE_BLOCK2
else:
    CODE_BLOCK3
```

Now here if CODE_BLOCK1 executes then it will execute the statements inside it. Which is another conditional.

Try out the following code on the interpreter

```
>>> if 1<10:
. . . .    print(“Boolean_Expression1 is true So I will be printed”)
. . . .    if True:
. . . .        print(“Boolean_expression1 is True and I am also True So I will also be printed as I was inside CODE
BLOCK 1”)
. . . .    else:
. . . .       print(“Because my corresponding Boolean Expression is True so I will not be printed”)
. . . .elif 2<10 :
. . . .    print(”Boolean_expression2 is true but Boolean_expression1 is already true so I will not be printed”)
. . . .else:
. . . .   print(“No one gives a damn about me”)
Boolean_Expression1is True So I will be printed
Boolean_expression1 is True and I am also True So I will also be printed as I was inside CODE BLOCK 1
```

Interview Questions

What do you mean by suite of statements?

Its essentially the same as a code block but the code block is referred to as suite of statements many times throughout the python documentation.

What do you understand by STDIN, STDOUT and STDERR?

These are standard data streams in and OS used for Taking Input in OS from terminal, Giving output to terminal by OS and Giving information about a fault that occured in a program to the OS correspondingly.

Where are stdin, stdout, stderr in *nix systems?

/dev/

What’s separator and end in print statement?

If we give print statement multiple objects to print then it prints those objects separated by the “sep” and it ends the whole content that it prints with “end”.

Is there ternary operator in python like condition?exp1:exp2

Yes.

But not with this syntax.

The syntax of Python’s ternary operator is-

```
exp_if_condition_true if condition else
exp_if_condition_false
```

```
Eg. >>> 1 if 1==1 else 2

     >>> 1
```

Give a real-life example when you feel nesting of conditionals?

```
if I_study:
    If I_study_whole_semester:
        I_will_score_85
    elif I_study_2_days_before_exam:
        I_will_score_80
    elif I_study_night_before_exam:
        I_will_score_60
    else:
        I_will_pass
else:
    if good_luck:
        I_will_pass
    else:
        I_will_fail
```