Python Basics

Sowmya Vajjala

10/8/2020

The goal of this tutorial is just to walk you through some basic Python syntax, and get you started with small programs. You can closely follow this or skim through this or skip entirely depending on your comfort level with Python. Considering that I will use Python examples later, I am starting with some basics.

I am using either Python console or the project interface in PyCharm. You are free to use what you want (Atom, Spyder, Mac/Unix terminal etc.).

Instead of copy pasting, try to type the code in this document. There will be a video accompanying this text - I strongly recommend using both together to get a better understanding!

1. Python Variables

Doing Basic Calculations in Python

```
2+3
```

```
## 5
```

```
2-3
```

```
## -1
```

```
2*3
```

```
## 6
```

```
2/3
```

```
## 0.6666666666666666
```

```
2%3
```

```
## 2
```

```
5//2
```

```
## 2
```

```
5**2
```

```
## 25
```

Variables and Assignments

```
a = 2
b = 4
c = d = 5
e = "python"
trial = 2.4
bunch1 = [1,44,5,53,1]
bunch2 = [1,22,33,3.55,1.5]
bunch3 = [2,1,"Python","Programming"]
#Note: space here is optional, added only for readability
```

Variable naming

- Spaces are not allowed in names
- Names should contain only letters, numbers and underscore
- Names cannot start with a number
- They are case-sensitive (a, A are not the same variable)
- Convention: start with a lowercase character.
- Convention: use underscore between words of a long variable name
- Variable name should not be one of the 'keywords' in Python

Re-assigning Variables

```
first = 1
second = 2
first = "Python3"
first, second=2.5,"Changed"
first
```

```
## 2.5
```

```
second
```

```
## 'Changed'
```

Try these out!

```
a,b,c = 2,3,4 #(What is this doing?)
a = b #(What is a, b, c after this?)
a == c #(What is this??)
d = hello #(Will this work?)
e #(what happens after this?)
```

Your First Python Program

- Go to PyCharm. In File Menu, choose "New Project" and name it EconNLP course (or something like that). A project in this context is a collection of programs you write.
- In this new project Week2, Rightclick and choose New->Python File. Name your file: MyFirstProgram or something.
- Go to MyFirstProgram.py and type: print('Hello World!')
- In the next line, type: print("This is my first Python program")

- Click on the green pennant/triangle to run/execute your program. (You can do all this in any editor you want and run the program)

Your second program

Write a program basics.py with the following lines.

```
print(4)
print(4*7)
print("hello " * 4)
print("hello" * 4)
x = 3
print(x)
print(x*4)
y = x
print(y)
name = input('Enter your name: ')
print ('Hello', name)
```

-Does it run? If it does not run, make it run. What do you see? It ran completely if you saw the message: "Process finished with exit code 0"

To err is human. To not forgive is computerish.

What happens when you type these on console:

```
print("name)
("name')
print("John's pen")
print('John's pen')
print('name')
```

What happens when you type:

```
x = "hello"
print(x+5)
```

Why?

What happens when you type:

```
print(3)
print(1,000,000)
```

Did python give you what you expected in both cases? Why? Why not?

Asking the user for input

input() is a built-in function in Python to seek input from user. (print is another function we already used so far. We will get to "what is a function?" soon.)

type: input() on your console. Press enter. What happened?

If you now type something on the console, and press enter, what happens?

```
i = input("Enter a number: ")
```

- type this on the console and see what happens. If you now type i and press enter, what does it print? Is that what you would expected to see?

Conversion between different data types

If we ask the user to enter a number, and it is read by default as string, what should we do? Python has some built in functions to convert between data types. For example, int() converts whatever you give it into a integer if possible, otherwise, it throws an error. Try the following on your console:

```
int("3")
int("3.4")
int("a")
```

similar functions float(), str() etc too exist.

Operators in Python

Operators are those symbols between variables that allow you to perform some "operations". mathematical operators are: +, -, /, *, % etc. Logical operators are and, or, not. Comparison operators are: ==, <, <=, >, >= (there are few more - more on those later). When a line of code has more than one operator, rules of precedence exist to decide what operation should be performed first.

What is the output of these two lines?

```
5-3*2
(5-3)*2
```

Rules of operator precedence

parenthesis come first - anything within parentheses gets executed first. Exponents come next.

```
2**1+1 #is 3, not 4
3*1**3 #is 3, not 27
```

Multiplication and division have the same level of precedence. Finally, addition and subtraction have same precedence. When you have two operators of same precedence, execution is left to write. Thus, 5-3+2 is 4. General advice: use parentheses, to avoid confusion.

Btw, # is used to comment in Python. Anything that comes after # in python code is commented out in that line. [Exercise for you: Check how to write multi line comments!]

Practice Question

Switch back to Python Project mode from console mode and create a new code file called SecondProgram and write a code that does these things: - Prompt the user to enter their name. - Prompt the user to enter their country. - Write a print statement which says: "Hello [Name] from [Country]. Welcome to Munich." - Run your program, and test with the following values: Name = Bond, Country = England.

You can try a few more exercises Chapter 2 of "Python for Everybody" or from any online textbooks such as this - go to the end to see the questions.

2. Python Conditionals

Let us now take a quick look at how to write conditional statements (e.g., if this, do that; if this and that, do these two etc.) in Python. Before doing that, we have to understand how to use logical (and/or/not) and boolean (true/false) operators in Python.

Boolean Expressions

What is a Boolean Expression? - Something that is either true or false, and where there is no other possibility.

Where are boolean expressions useful? - When you want to compare two things and make a decision based on the true/false situation.

Boolean expressions in Python:

```
x == y, x!= y #(to check if x and y have the same value)
x > y, x >= y
x < y, x <= y
x is y #(x is the same as y)
x is not y #(x is not the same as y)
```

Keywords appear in blue in this above example.

Two questions for you: - What is the difference between = and == symbols? - What is the difference between == and *is**?

Try these out in the python console, and guess the output (true or false?) before seeing it:

```
5 >= 4.9999999
5 <= "5"
a = "string"
b = "string"
a==b
x = 3
x==3
"tail" < "trail"
"tango" < "mango"
2 > 5
```

What surprised you here? Figure out why!

Logical Operators

Three logical operators you should know: and, or, not. Using logical operators between two expressions results in a boolean value. - **and** : (expression X) and (expression Y) is true only if both the expressions are true. - **or**: expression X) or (expression Y) is true if any of the expressions is true. - **not**: not(expression X) is true if expression X is false.

Conditional Statements

Conditional statements allow decision making in a program. They are typically of the form: if(A), do B. Else, do C. They can be as simple as a single if-else or a chain of conditions or conditions within conditions and so on.

A simple conditional statement:

```
x = 5
if x==1:
   print("x is one")
else:
   print("x is anything except one")
```

Note the indentation in these examples. Python is heavily focused on indentation. So, stuff within a condition definition should all align to the same indent level. Same with everything else we will see.

A chained conditional statement:

```
x = 8
if x==0:
   print("x is zero")
elif x==1:
   print("x is one")
else:
   print("x is neither zero nor one")
```

A question: what happens if x was "blah" or 2.5 in this example? Will there be a error? Why? Why not?

Nested Conditional statement

```
x = 2
y = 5
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

Again, what happens if x is "a" and y is 5? Why?

Practice Problems

1. Write a program that asks the user to enter a temparature in Celsius, and prints the temparature in Fahrenheit (Textbook question).
2. Write a program that asks the user to enter a temparature in Fahrenheit and prints the temparature in Celsius.

Hint 1: Celsius to Fahrenheit conversion: (user_input*9/5)+32 = your answer.

Hint 2: Fahrenheit to Celsius conversion: (user_input-32)*5/9 = your answer.

(You can always send an email to our mailing list with your queries - anyone who knows the answer can respond!)

3. Writing Functions in Python

Some functionalities are already implemented in Python, we don't need to write our code, we can just **call** these functions in our program. Some examples are: print(), int(), input() etc. These are all built in functions.

There is more built in code in Python we can just use. **Modules** are a collection of several functions put into a large python file. We can **call** a module into our code by using **import** keyword followed by the module name.

Exercise: Apart from functions and modules, you will also see the word **method** being used - Try to find out the difference between these two!

Why do we need to write our own functions? Python already seems to have so many of them! There are two reasons: - To implement our own custom sequence of programming events - To avoid repetition. We called print() so many times today, but we did not write the entire print() function everywhere.

Some basics: - A function needs to be always first defined, before being called. - A function definition starts with a def keyword. - A function need not necessary have a return statement, but where you can use return instead of print, use it. - Indentation and syntax needs to be strictly followed even with functions.

Take this example from "Python for Everybody" textbook (Chapter 4), which illustrates the use of functions:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

def repeat_lyrics():
    print_lyrics() #-this is a function "call"
    print_lyrics() #-this is a function "call"

repeat_lyrics()  #-this is a function "call"
```

What is the sequence of statements in this? What gets executed first? - Functions have to be defined before they are "called". - Program execution starts from the first statement of a program (or, if there is a "Main" function defined. More on this later) - When Python sees a function call, it takes a detour, goes to that function, runs through that, and "returns" to where it stopped earlier.

Arguments and Parameters for Functions

```
def add_str_int(some_string, some_int):
   print(some_string, str(some_int))
#some_string, some_int are parameters.
add_str_int("python",3) #"python", 3 - are arguments.
add_str_int("Week", 4) #"Week", 4 - are arguments
```

Fruitful function and void function

Fruitful functions - give you back something as output, which you assign to some variable in your program.

```
def sum_two_numbers(a,b):
   return a+b #return is a keyword that "gives back" the function's result to us.

c = sum_two_numbers(5,6)
d = c**3
print(d)
```

Void functions don't "return" anything.

```
def sum_two_numbers(a,b):
   a = a+1
   b = b+1
   print(a + b)
c = sum_two_numbers(2,3)
print(c)
```

Errors and error handling in code

In Python, we use a kind of conditional statement called <try,except> to do error handling. It is like an if-else, but serves a different purpose - used when we expect possible errors that may occur (e.g., division by 0, wrong input etc)

```
try:
   number = input("Enter a number: ")
   next_number = int(number)+1
   print(int(number)/next_number)
except Exception as e:
   print(e)
```

What is the difference between using a try-except block and not using it and just running the three lines of code?

```
number = input("Enter a number: ")
next_number = int(number)+1
print(int(number)/next_number)
```

Practice Exercise

Read this program description: It has two functions: ctof, ftoc - celsius to F, F to celsius (use what you wrote earlier!). Prompt the user to choose a temparature scale: C or F. If something else is chosen, you should stop the program there, with a message suggesting them to type either C or F. If the user entered C, call CtoF, print the output. If they chose F, call FtoC, print the output.

4. Loops and Iterations in Python

We use loops to repeat the same thing again and again (e.g., keep asking the user for input until they want to stop). [Question: What is the difference between a loop and a function, then?]

A "while" loop

```
i=0
while i<3:
    print(i)
    i = i+1
print("Done with the while loop!")
```

A "for" loop

```
i=0
for i in range(0,3):
    print(i)
    i=i+1
print("Done with the for loop!")
```

There are different ways of writing these loops. We will see them as we progress, or you can take a look at the textbooks for a detailed discussion.

An example program with both while and for loops

```
def someFunction(number):
    result = 1
    i = 1
    while i<=number:
        result = result*i
        i = i+1
    return result
print(someFunction(5))

def someFunction2(number):
    result = 1
    for i in range(1,number+1):
        result = result*i
    return result
print(someFunction2(5))
```

Infinite loops!

```
tempstring = "whatever"
while tempstring == "whatever":
    print(tempstring)
print("You will never see this message")

#Another one:
while True:
   print("I won't stop!")
print("You will never see this message")
```

- these kind of loops will never stop until you apply force (on your keyboard, that is). (why??)

break and continue statements

Suppose you want to keep taking input from the user until the user enters "done". **break** statement is useful here.

```
while True:
   line = input('Enter something: ')
   if line == 'done':
```

```
        print("Stopping here")
        break #break statement breaks the loop.
    else:
        print(line)
```

break lets you out of the loop completely. **continue** just comes out of current iteration and goes to the next iteration of the loop.

```
while True:
    line = input("Enter something: ")
    if line == 'pass':
        print("I am passing without printing what you entered")
        continue
    elif line == 'done':
        print("I am stopping the program.")
        break
    else:
        print(line)
print("Done!")
```

Question for you: How do you choose between for and while?

Exercise: Spot the bug

```
def someFunction(number):
    result = 1
    i = 1
    while i<=number:
        result = result*i
        number = number+1
    return result
print(someFunction(5))
```

Exercise: Analyze This!

```
def seq3np1(n):
    while n != 1:
        print(n, end=", ")
        if n % 2 == 0:        # n is even
            n = n // 2
        else:                 # n is odd
            n = n * 3 + 1
    print(n, end=".\n")
```

What will seq3np1(16) print? why?

Exercise: Write a program that follows this process:

```
> Enter the number of numbers you want to enter:
5
```

```
> Enter a number: 2
> Enter a number: 6
> Enter a number: 5
> Enter a number: 3
> Enter a number: 8
> The sum of these numbers is: 24
> The average of these numbers is: 4.8
```

May be write a program that takes a number and prints multiplication table for that number (n*1 to n*10, one number per line)? You can practice through the questions at the end of a chapter on loops in any Python textbook!

5. Reading and Writing Files

There are three basic operations involved: opening a file, reading its contents, and writing content into a new or existing file. We will be talking primarily about files with text content in this course. (and in some plain text format for now, other formats in later videos!)

You open a file with open() function. open("/home/Desktop/a.txt") creates a "handle" for the file if the file really exists on your computer. Otherwise, it throws an error.

Reading a file's contents

I am using a file mbox-short.txt, taken from "Python for Everybody", Chapter 7. Assuming that I have this file in my code folder, here is how to read it:

```
#Reads line by line
hand = open('mbox-short.txt')
content = ""
for line in fhand:
  content = content + line
print(content)

# Reads all lines at once:
fhand = open('mbox-short.txt')
content = fhand.read()
```

[Question:] How do you check for the exact file path on your computer to tell Python? How can you check if a file exists?

Writing into a file

```
writeHandle = open('output.txt', 'w')
writeHandle.write("this is a line\n") #\n is newline
writeHandle.close()
```

Try to find out how to write a program to get a listing of all files in a folder given by the user.

6. Back to functions and main function

Generally, it is a good idea to encapsulate your program into a collection of functions, and define a **main** function as a sort of entry point to the program, which calls other functions. (see here for more advanced discssion)

Here is an example program with a main function.

```
def factorial(n):
   fact = 1
   for i in range(1,n+1): #why not start at 0??
      fact = fact*i
   return fact
def main():
   num = int(input("Enter a number: "))
   print(factorial(num))
if __name__ == "__main__":
    main()
```

[Question:] What does that line before main() call mean?

Exercise

Write a program with the following functions: 1. OddEven(integer): This function takes a positive whole number as an argument, and returns a string which is either "Odd" or "Even". 2. LogNum(integer): Takes a positive whole number and returns the logarithm of this number. 3. RandNum(integer): Takes a positive whole number and returns a random number between 0 and this number. 4. main(): A main function, that prompts a user for a number, and returns the output of all the above functions one by one. Make sure your program actually runs! Note: program should ask for input only once, and give that number as argument to all functions!

With this, I will end the "Python Basics" part. I will prepare another similar tutorial on "Python Data Structures", to wrap up Python fundamentals. Look for accompanying videos - pair this text and the video together, and do the exercises on your laptop!

Please note that since this is not a Python course, we have to do a super fast introduction (and virtually, at that!). I would strongly recommend you to spend more time with a textbook, practice code exercises, and follow a Python class if possible, if you feel you need to know more!