

Python, an introduction

Samarth Ramesh

DRAFT

Draft

Chapter 1

Hello World

Python is a programming language, developed by the Python Software Foundation and released under the PSFL License. I quote Wikipedia:

‘Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python’s design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

For those who have never programmed in their lives, Python offers an easy in of sorts, as the language is very easy to both read and use.

1.1 Python installed?

To check if you have Python already installed, open the command prompt and type

```
Python
```

If Python is installed, the command prompt should reply

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC  
v.1924 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more infor-  
mation.  
>>>
```

However, if you don’t see a message like above, but instead see:

'python' is not recognized as an internal or external command,
operable program or batch file.

then, you don't have a copy of Python on your computer. To install Python, go to www.python.org/downloads/ and follow the instructions. Now that all of us have Python installed on our computers, we may begin our exploration of the language.

1.2 First program

Finally, our first program. Our aim is to print the statement "Hello World" (Interestingly, the tradition began with Ritchie & Kerningham's book 'The C Programming Language')

```
print("Hello World!")
```

Would simply return:

```
Hello World!
```

In Python 3, the statement *print()* was used to print a statement, which in the above case was a string which was enclosed in "double quotes".

1.3 More Printing

If we so desire, we may also print multiple strings at the same time using the *print()* function as shown:

```
print("Hello World!","Hoorah!!")
```

which would return

```
Hello World! Hoorah!!
```

The *print()* function inserts a newline after printing the arguments passed to it. So in other words,

```
print("Hello World!","Hoorah!!")
```

and

```
print("Hello World!")  
print("Hoorah!!")
```

Would return two different outputs as shown

```
Hello World! Hoorah!!
```

and

```
Hello World!  
Hoorah!!
```

respectively.

To force the printing of a newline, we use `"\n"` in our string as shown.

```
print("Hello World! \n Hoorah!!")
```

The above script returns the same output as:

```
print("Hello World!")  
print("Hoorah!!")
```

But the former is preferred for the sake of brevity.

In a similar manner, we may also format output with `"\t"` to force a new tab, and `"\r"` to force the cursor to return to the start of the line.

It is useful to know that `"\"` character is also called an "Escape Character", that converts a special(reserved) symbol into an ordinary character.

1.4 Variable Variables

Now that we have started using the `print()` function, we may further explore it.

the print function can also print the value of variables.

```
somevariable="Hello World, I am a variable"  
print(somevariable)
```

returns

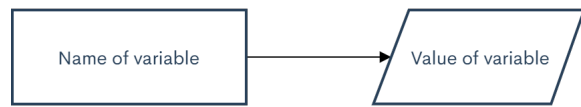
```
Hello World, I am a variable
```

The variable can be reset to another value if needed.

In short, a single 'equals' mark indicates assignment of a variable. The entire process can be described as:

In Python, all variables are not *declared*, they are *assigned*. This highlights the fact that Python is Dynamically typed.

Dynamically typed means that the type of a variable is checked during runtime, and not during *declaration*.



We give a value to a variable when we initialize it.

NB: A language is statically-typed if the type of a variable is known at compile-time instead of at run-time. A more comprehensive list of differences can be found in Appendix A

We may also print multiple variables and/or strings at the same time, as with strings, in a similar fashion. For example:

```
var1="abc"
var2="def"
print(var1,var2,"ghi")
```

Which would return

```
abc def ghi
```

1.5 Arithmetic

Python, like most programming languages has some basic math in its Standard Library. The four symbols, '+', '-', '*', and '/' are used with both variables and constants alike.

NB: A more comprehensive list of mathematical functions (among others) can be found at the end of the book in Appendix B

For example to find and print the product of 256 and 456, we can use:

```
a=256
b=456
c=a*b
print(c)
```

The *print()* function takes both variables and strings as inputs and can concatenate the values of both in place. thus, to give more clarity to the end

user, a statement can also be included. For example, to calculate the value of 234×235 and give a statement about the same, the following two scripts may be used

Draft

1.6 Exercise

1. Write a program to convert *Celsius* to *Fahrenheit* for the given temperatures $t = 45^{\circ}\text{C}$ and $t = -40^{\circ}\text{C}$.
2. Write a program that calculates the average of three numbers 189086, 127809, 1567801.

Draft

Chapter 2

Control Flow

So far, our programs have dealt with direct statements and have not had to make any decisions. The backbone of most languages is the *if* statement. In the first chapter, while introducing python, I had quoted wikipedia and wrote:

Python's design philosophy emphasizes code readability with its notable use of significant whitespace.

in other words, every whitespace, that is a group characters representing horizontal or vertical space, for the purpose of improving readability and in the case of Python demarcating blocks of code, is, an indentation, a space or a newline. The use of indentation, while optional in many other languages, is as a result, compulsory in Python. The beginning of a code block is marked by an increase in indentation and the end with the decrease in indentation.

2.1 The *if* statement

The *if*-statement has the following syntax

```
if condition :  
    effect
```

Note the use of a colon at the end of the condition. It is what instructs the compiler to look at the next indentation.

For example, the following script tests if two given numbers are equal or not.

```
a=25  
b=30  
if a==b:  
    print("a=b")
```

However, the *if* conditional also includes support for *else* with a similar syntax.

So a program to check if a number is greater than, lesser than or equal to another number, we can write:

```
a=17890 b=12908 if a>b:
    print("a > b")
else:
    if a<b:
        print("a<b")
    else:
        print("a=b")
```

For the sake of brevity, Python also supports *elif* statement that allow us to say "if the above statement was false, then see if this statement is true" thus, the above script can be rewritten as

```
if a>b:
    print(a > b)
elif a<b:
    print(a < b)
else:
    print(a = b)
```

2.2 *For* loops

The *for* loop is widely regarded as the workhorse of most languages and is used to execute a certain block of code multiple times.

Like the *if* conditional, the *for* loop also needs a new level of indentation. The loop, uses the following syntax

```
for somevariable in iterable : #or sequence
    do this
after exiting do this
```

The 'something' initially is generally `range(somethingelse)`, where 'somethingelse' is a number. The range function by default starts counting from 0. so a script to calculate the various values in fahrenheit of temperatures in celsius starting from 10° to 100° with increments of 10 would be

```
x=0          #initializes a variable x to 0 to be safe
for x in range(9):      #we use 10, while 0-10 includes 11 num-
    bers, the range excludes the last number
```

```
celsius=(10*(x+1))    #getting the celsius from numbers
fahr=((1.8*celsius)+32)    #converting celsius to fahr
print(celsius, "degrees C =" , fahr , "degrees F")
```

The script if entered from the command prompt would take time to enter in one by one, and often tends to be more trouble than it is worth. However, if you used some text editor or IDE, then the program would have run.

So far, we have dealt only with simple and basic programs that did not have many loops and conditionals. However, as our journey progresses, the complexity will only grow. Until now, we have started Python from the command prompt using the *python* command. Henceforth, it is recommended that you use an IDE or something like Notepad++ (for Windows), save the script as something.py, then from the command prompt, cd to the working directory and finally pass the filename as an argument to Python.

In other words, follow the given steps:

1. First, open the command prompt by searching for 'cmd'
2. Then, enter the command 'mkdir python'
3. And next enter the command 'cd python'
4. Then in Notepad++, type and write your script, and save it into the folder you just created
5. Then, if the name of the script is say script.py, the type "python script.py"

2.3 Exercises

1. Create a program to write the multiplication tables of the first 10 numbers, upto 10 (that is, the program must return the times tables from $x*1$ to $x*10$, for all x between 1 and 10, both included)
2. Create a general program to calculate the average of 'n' numbers, where every third natural number upto 25 is taken. (that is, if later on the limit increases, the program remains the same)

Draft