Python Lesson 1

Derek Chen

September 18, 2023

1 The Primitive Types

In terms of popularity, Python is easily a top-three language (alongside Java and C++). Syntatically, it is definitely one of the easier languages to learn. Unlike Java, C++, or any other standard programming language, it disregards brackets ({}) and substitutes them for indentation. Additionally, the developers for Python designed the language to be as similar to English as possible and tried to keep syntax as concise as possible.

One way Python does this, is allowing the computer to identify variable types. For example, we know that if I type into my text-editor:

$$x = 6$$

We know that Python has created a variable "x" that holds the value 6. But there is a lot more going on underneath the hood of the computer.

In all programming languages, there are variables types known as "primitive types". While there are multiple primitive types, the ones we will be concerned with for the most part are *int, char, boolean, double, float, long* (with strong emphasis on int, char, boolean, double).

- *int* = Integer (6)
- *char* = Character ('c')
- *boolean* = Boolean (Truth Values) (True)
- *double* = Numbers with decimals (6.25)
- *float* = Numbers with decimals (less precise) (7.15)
- *long* = Integer (with wider range) (4)

Fortunately for any Python developer, you'll usually never have to worry about assigning your variable types. But it is important to understand what we can and cannot assign to variables.

2 Basic Operations

Now that we know what the primitive types are, let's take a look at some operations we can use on these primitive types.

- + → Addition
- - → Subtraction
- * → Multiplication
- / → Division
- % → Modulo (Remainder)

In most programming languages, you cannot perform operations on two different primitive types without converting one (i.e multplying an int and double). However, Python does the heavy-lifting and you can freely operate between differing types.

3 Advanced Operations and Conditionals

However, we can do more math besides the basic operations. In Python and any other programming language, we have access to other basic math symbols.

- $>$ → Greater Than
- $<$ → Less Than
- $\geq$ → Greater Than or Equal To
- $\leq$ → Less Than or Equal To
- $=$ → Equal To

To test if inequalities or equation are true, we can use *conditional statements*.

if(x == 6): print("True")

else: return("False")

If we want test multiple statements at once, we can join conditional statements together using the "or"/"and" syntax.

or symbol

if(x == 6 || x == 7): print("True")

else: return("False")

and symbol

if(x < 6 && x > 3): print("True")

else: return("False")

4 Non-Primitive Types

In any programming language, everything is built off of the primitive types. Any type that is not primitive, we will classify as non-primitive types. The most obvious examples are arrays and strings.

I'll tend to lump arrays and strings together, as strings are just character arrays. Arrays are usually multiple instances of primitive type next to each other. Again, Python does the heavy lifting and you won't have to specify that your variable is an array or String.

str = "Hello World"

birthday = [8, 2, 04]

Both are valid ways to declare a string or array

4.1 Length of the Array

A common feature that most programming languages give their developers is the ability to know the length of the array they have. Using the birthday example, we can use the following command:

print(len(birthday))

And this will display the length of the array of birthday, (which is 3).

4.2 Loops and Iteration Through Arrays

As an example, let's say you are an owner of game store. It is important to know who is working at your store. Let's say we had an array of 6-digit integers, and this was a list of hypothetical worker IDs. We will define the array to be:

IDs = [858386, 923935, 230596, 034920, 350693]

Let's say you wanted to check if an ID was in your system. What Python code could you write to check an ID was in the array of IDs?

By laws of hypothetical science, we can only know if something DOES NOT exist if we look everywhere. For example, if you lost your car keys, you would have to look through your entire house to determine that your car keys are missing.

It's no different in this example. We have to scan through the entire array, and the best way to do that is check every value in our array. The best way to do this is using what we call a loop.

There are two different types of loop that we will be concerned will for now. The *while loop* and *for loop*. The while loop iterates through based off a conditional. As long as the conditional is met, the while loop will keep looping. For example:

index = 0

while(index < len(IDs)):

This is ok. The for loop is much better for this problem we have. The for loop iterates based on a specific number of loops. We know how large our ID array is (thanks to the len() command). Thus let's view what a for loop would look like.

for x in IDs:

if(ID == IDweAreLookingFor): print("Found")

Note all for loops are while loops. Everything you can accomplish with a for loop can also be done with a while loop (usually). The general rule of thumb is use while loops when you don't know how many times you will iterate and use for loops when you do know.

4.3 Adding and Removing From the List

Suppose your game store hires a new employee. You need to add a new ID number to your list of IDs. Usually in most programming languages, you'll have to create a new array, store all the existing values, and add the new value.

However, a feature that is unique to Python (as opposed to most standard programing languages) is the ability of dynamic allocation. In other words, Python allows you to actively add new values without creating a new array (it's all about making your life easy!).

Suppose we had a new employee named John and his ID was 835970. To add his ID to our list, we do:

IDs.append(835970)

this adds the new ID at the end of our list

What if an employee quit? We don't want to keep IDs that are not in service. Similarly, we can remove from the list.

IDs.remove(835970)

this deletes the value 835970

Alternatively, if you want to remove at a specified index, you can do:

IDs.pop(1)

this deletes the SECOND element in the IDs list

But why does pop(1) remove the second element? This is a good segway to the topic of *computer science counting*. Let's reuse our IDs example back on page 3. When we created the array, there was 5 IDs in the array (ignore John the new employee). However, the first ID is located at the 0th index. So, the list would look like:

IDs = [858386, 923935, 230596, 034920, 350693]

- 858386 - 0th index
- 923935 - 1st index
- 230596 - 2nd index
- 034920 - 3rd index
- 350693 - 4th index

In other words, the big idea is that we start at 0 whenever we count. This is big whenever we use loops and arrays.

4.4 Retrieving Elements

The powerful thing about arrays is this idea of *randomized access memory*. It is a long word and we won't get too bogged down in the details, but the main idea is we always have access to every item in our arrays. For example, let's say we wanted to know who the first employee was at our store. It would obviously be whoever's ID is first in the array. Using the concept of computer science counting, we can say

print(IDs[0])

and that will get the first element in our array.

5 Building Logic

We have gone over a LOT of syntax and ideas. It is important to always know that you are not ever expected to memorize syntax at the beginning. It is ok when you learn new ideas to google them and learn more in-depth about syntax.

Another difficult aspect of coding is building logic together. With the tools I have explained right now, it might seem like you can only perform basic mathematical operations. However, as we build our skills up, we will eventually

reach the point of building common logical operations that computer application might typically use (i.e identifying a strong password).

This homework assignment is designed to get you to practice utilizing the skills you have just learned. Coding does not have a specific road map to its problems, so it is up to you to creatively design solutions to problems.