The Python Book

By Kalu Kalu

Table of Contents

Careers With Python

Software Developer/Python Developer

A Python Web Developer is responsible for writing server-side web application logic. Python web developers usually develop back-end components, connect the application with the other (often third-party) web services, and support the front-end developers by integrating their work with the Python application.

Data Analyst

Data analysts translate numbers into plain English. Every business collects data, whether it's sales figures, market research, logistics, or transportation costs. A data analyst's job is to take that data and use it to help companies make better business decisions.

Data Scientist

A Data Scientist combines statistical and machine learning techniques with Python programming to analyze and interpret complex data. (datacamp.org)

Quantitative Analyst

In finance, quantitative analysts ensure portfolios are risk balanced, help find new trading opportunities, and evaluate asset prices using mathematical models.

Running Python Programs

Two Ways To Run Python Programs

There are two ways to run python programs. One way is by typing your python commands directly into the python interactive interpreter of your terminal. Another way is to first use a text editor to type your python into a python “script” or “program” and run that “script” (aka program). We will go through each way in turn.

Method 1: Using The Python Interactive Interpreter

As mentioned above the first way to type python code is by using the python interactive interpreter. The python interactive interpreter aka your “python shell” is a way to quickly test

out python code. You type and run commands, generally one line at a time. Please open up your Terminal and enter python3 by typing “python3” in your terminal.

Pic1

If you see a command that says python3 is not available, then simply try python, like pictured below.

Pic2

Once you are in your Python Interactive Interpreter you can try your first bit of python code. Please type the following:

```
>>> bill = 54.87
>>> tip_percent = 0.10
>>> tip = bill * tip_percent
>>> total = tip + bill
>>> print("Here is your total tip amount")
>>> print(tip)
>>> print("Here is your total bill")
>>> print(total)
```

Your output should look like this:

```
Here is your total tip amount
5.487
Here is your total bill
60.357
```

Pic3

Congratulations. You have just ran your first code using the Python Interactive Interpreter. To leave the interactive interpreter and get back to your bash shell please enter the command quit()

```
>>> quit()
```

Pic4

You should now be back in your Bash Shell.

Method 2: Saving Your Code In a Script And Running The Script

The second method of running python code is using a Text Editor, such as Sublime, to type your code in a script and then saving and running that script in your terminal.

Creating A Python Script In Sublime

To get started first open a new sublime document. Then click File > Save As

Pic5

We want to save the Python file somewhere where we will remember it when we wish to run it. And we want to make sure we end our filename with .py so that the computer knows it's a Python program. Why don't you name the program, calculate_bill.py . Make sure that you do not use a space when naming python programs. You can use an underscore, _ , instead of a space.

Remember, we want to pay special attention to **WHERE** we save it. For now just save it in the Desktop.

Pic6

With the file saved, we can move on to typing our code in our newly created calculate_bill.py file. Inside of your sublime calculate_bill.py please type the following:

calculate_bill.py

```
bill = 54.87
tip_percent = 0.10
tip = bill * tip_percent
total = tip + bill
print("Here is your total tip amount")
print(tip)
print("Here is your total bill")
print(total)
```

Pic7

Saving Changes In Sublime

And then save. You will know when your file is saved in Sublime, by look at the file tab. If there is a grey dot next to your filename, then you have not yet saved. If there is a grey X instead then you have saved all changes.

When it is unsaved it looks like this: Grey Dot

Pic8

When it is saved it looks like this: Grey X

Pic 9

Running Your Script In The Terminal

Now that we have our created our script it is time to run it. Please open your Terminal.

Pic10

First, we must “cd” into the directory (aka folder) where we saved our calculate_bill.py file. Since we saved it in our Desktop, which is located in our home folder, we need to cd into ~/Desktop .

```
Kalus-MacBook-Pro:~ kalukalu$ cd ~/Desktop/
```

Pic11

Once we are in the same folder as our script, we can run the file using the python keyword and the name of our file as an argument. Python calculate_bill.py

```
Kalus-MacBook-Pro:Desktop kalukalu$ python calculate_bill.py
```

We should get the following output printed on our Terminal screen.

```
Here is your total tip amount
5.487
Here is your total bill
60.357
```

See pic12

Do Not Confuse Your Bash Shell With Your Python Shell

Many students make the mistake of confusing their Python Interactive Interpreter (aka Python Shell) with the Bash Shell. Your Bash Shell is the default shell that your Terminal opens up in. Bash is a programming language. With this programming language you can direct the operation of the computer by entering commands as text for a command line interpreter to execute.

Python is a completely different programming language with different grammar and syntax. You cannot enter python code directly inside of a Bash Shell. You first must use some Bash Command to enter inside of your Python Shell. Once in your python shell you can comfortably enter Python Code directly into the shell. To quit your Python shell and return back to the default Bash shell, you simply type the command quit().

Python Shell or Python Script?

Great job on running your first python code. From here on out we can type our python code in a script file, save that script file and then run that script file from our Bash shell.

Import Python Scripts/Modules

An important part of programing is importing program that other people (or yourself) wrote. This helps with the maintainability and readability of code, since you can break up your python code into different script files based on shared features, make your code more “modular.” Another important benefit of importing scripts that cannot be overstated, is that this allows you to use other people’s programs. In fact, all of program is importing one script that does a

certain feature. We will learn more about this later. For now let us learn the simplest form of importing a python script .

Three Ways Of Importing

There are three basic ways of importing objects from a module that we will go over in further detail below. These three ways are 1) Importing the objects from a module one at a time, importing all the objects from a module all at once using a wildcard, importing the entire module itself. We will go through each part below.

Importing A Script File

To get started, let's create two modules. A module is just another name for a python script. We will create our modules in our Desktop. We should already have a file called calculate_bill.py in your Desktop. Create two other files in our Desktop, one called data.py, and another called main.py. Please make sure that you create these two files, main.py and data.py, along side the already existing calculate_bill.py .

We have created our modules! Now time to write some code inside of them. Let's create some dummy data inside of main.py. We will start with very simple data. No need to worry if you do not understand the syntax of creating variables for now. We will go over that in the later chapters.

data.py

```
name1 = 'Jon'
name2 = 'Barry'
name3 = 'Eke'
name4 = 'Uche'

age1 = 24
age2 = 45
age3 = 62
age4 = 14
```

Import Objects One At A Time

Next inside of main.py we will import the data that we created in data.py. Please type the following code inside of main.py

main.py

```
from data import name1, age1

print(name1)
print(age1)
```

As you can see the syntax for importing objects is: from filename import object_name. When writing the filename, you do not need to include the .py extension because Python already knows it's a python script. That's why we do "from data" and not "from data.py".

Now that we have done our imports let's see if it works. Open your Terminal, cd into your Desktop and run your main.py python.

```
Kalus-MacBook-Pro:~ kalukalu$ cd ~/Desktop/
Kalus-MacBook-Pro:Desktop kalukalu$ python main.py
```

When we run the code, we should get the following output.

```
Jon
24
```

We have successfully imported and used our name1 and age1 variables from data.py inside of main.py. Great! But what if we wanted to print name2 or age2? Well, for every object we wanted to use inside of main.py we would have to import. If we tried to use name2 without first importing it, we would get an error.

But wouldn't it be exhaustive to import EVERY single object? In this case we would have to do from data import name1, name2, name3, name4, age1, age2, age3, age4 . But this can get much worse since we could be importing from a file with HUNDREDS of objects that we need. No one surely has the time to import potentially a hundred one by one? Of course not! We're programmers and we're lazy. We'll learn a better way next!

Import All The Objects At Once

As you might have guessed there's a way to just say import ALL the objects. That way is simply using a * inside of specifying the filenames to import. Therefore we need to change from data import name1, age1 to from data import * . Please change data.py as shown below to use the new wildcard import that gets ALL the object.

main.py

```
from data import *

print(name1, name2, name3, name4, age1, age2, age3, age4)
```

Then you can run your main.py script again in the Terminal.

```
Kalus-MacBook-Pro:Desktop kalukalu$ python main.py
```

Your output should like this:

```
('Jon', 'Barry', 'Eke', 'Uche', 24, 45, 62, 14)
```

As you can see we now have access to all the objects in the data.py merely by importing using the * wildcard. But there's one reason why you might not want to use this method. Sometimes you don't want ALL the objects of a module imported at once due to memory or other reasons. Yet you still want access to the objects of a module without having to import them one by one.

We have one more option for you.

Importing The Entire Module

So our last object is just to import the module, or script file itself as an object, and access each individual object in the module using dot notation. Let's write out some code to see how it works. Please change main.py to look like the following:

main.py

```
import data


print(data.name3, data.age3)
```

And now as as usual let's run it in our Terminal to make sure it still works.

```
Kalus-MacBook-Pro:Desktop kalukalu$ python main.py
```

Your output should like this:

```
('Eke', 62)
```

The if __name__ == "__main__" Use-Case

Time for a little experiment. What if we imported from our calculate_bill.py. Please change main.py to import and print the bill from calculate_bill.py.

main.py

```
from calculate_bill import bill


print(bill)
```

And run it in your Terminal.

```
Kalus-MacBook-Pro:Desktop kalukalu$ python main.py
```

Here is the output you will get, perhaps it will surprise you:

```
Here is your total tip amount
```

```
5.487
Here is your total bill
60.357
54.87
```

So, why did we get all the extra prints from calculated printed when we ran main.py? That's because of the way python works. In order for an object to be ran, EVERY line of code in the script is first ran.

But what if we have code that we don't want ran when import. Thankfully, there is a way to specify code that you ONLY want ran when being ran directly, not when being ran by another file another file that imported it.

Let's show you how to do this by editing our main.py to add an if statement. If you do not understand what an if statement is, for now don't worry. It enough to just copy and paste this code and for now understand, that if you put any code inside of this if statement, it will only run when ran directly.

calculate_bill.py

```
bill = 54.87
tip_percent = 0.10
tip = bill * tip_percent
total = tip + bill

if __name__ == "__main__":
        print("Here is your total tip amount")
        print(tip)
        print("Here is your total bill")
        print(total)
```

And now let's run main.py again.

```
Kalus-MacBook-Pro:Desktop kalukalu$ python main.py
```

Now only the information we wanted printed in main.py shows up.

```
54.87
```

But if we run the calculate_bill.py directly it will still run the print statements inside of our if statement.

```
Kalus-MacBook-Pro:Desktop kalukalu$ python calculate_bill.py
```

We can see that we get the calculate_bill print statements this time because we are running the program directly:

```
Here is your total tip amount
5.487
Here is your total bill
60.357
```

Basically, as we wrote it the if statement says, ‘only run the following lines of code if this module is ran directly’ (as opposed to imported).

If this confuses don’t worry about it. When it comes to if __name__ == “__main__” use-case, it’s likely that you’ll know when you’ll need it.

Variables & Objects

Let’s learn about the most BASIC building block of python. An object.

What is An Object?

Everything in python is an object. The first thing to know that that there are different type of object types, or data types. I discuss each important Python Data Type in the chapter below called Python Data Types.

For now, we must know that the MOST BASIC data type that all other objects are built on is the “object” type. For now all you need to know is that EVERYTHING in python is an object.

Creating Objects

Let’s create an object. For this example we will create a “string” object. Different objects are created with different ways. I discuss each object type below as well as the ways to create each object type.

For now, it’s important to know that specific objects have specific syntax used to create it.

Let’s create a string object. To do this we will quickly enter into our Python Shell. Open your terminal and type python3 to enter your Python shell. As will be discussed further in the String section under the Python Data Types chapter, a string is created by wrapping quotation marks around text. See the following.

Once your python shell is activated please type the following. This will create a string object.

```
>>> “Kalu Kalu"
```

We have just created a string object. When you create an object inside of your Python Shell, your shell will print the “string” representation of the object, or in other words, how that object is referred. For string objects, the string representation of the object is simply the text of the string. Therefore creating a string object as done above in your Python Shell will print the following output.

```
>>> “Kalu Kalu"
‘Kalu Kalu'
```

Pic13

Python Variables

We have created an object, and it has printed the string representation onto our python shell. Each object that is created is saved in computer memory by Python. If we want to have access to that object, we can give a name to the location in memory where it is saved. This is called a “variable assignment.”

Let’s create a another string containing “Kalu Kalu” but this time we will assign a variable name to it that we can refer to later when we want to access our string object. To assign an object to a variable name we must use, = , the assignment operator.

```
>>> name = "Kalu Kalu”
```

While we’re at it, let’s create another object, this time an Integer object. An integer object is created by simply typing a number (with no quotation marks). This time we will directly create and assign the object to a variable name.

```
>>> age = 29
```

Printing Objects

You can use the print() function to print an object onto the string. Let’s print our name and age objects on the string.

```
>>> print(age)
29
>>> print(name)
Kalu Kalu
```

Pic14

An Object Statement

When you are in the python shell, if you merely type the object as a command and press enter it usually (but not always) will show you the string representation of the object, similar to print.

```
>>> name
'Kalu Kalu'
>>> age
29
```

Pic15

Copying Objects

We can copy an object by assigning it to a new variable.

```
>>> nickname = name
```

Here we copy the name object, which contains the string “Kalu Kalu”. If we print nickname we will see that it now contains the string “Kalu Kalu”.

```
>>> print(nickname)
Kalu Kalu
```

Exactly the same as our name object

```
>>> print(name)
Kalu Kalu
```

Here we copy the name object, which contains the string “Kalu Kalu”. If we print nickname we will see that it now contains the string “Kalu Kalu”

Overriding Variables

We can override the objects stored in a variable name by simply making another assignment using the same variable name. For instance, let’s change our nickname to be a real nick name instead of just a copy of my name.

```
>>> nickname = "Kalu Times 2”
```

Now when we print nickname, it shows us our new string value, “Kalu Times 2”. The old value stored in nickname, “Kalu Kalu” is overridden no no longer exists.

```
>>> print(nickname)
Kalu Times 2
```

Variable Assignment Creates A Copy, Not a Reference

But what about our original “Name” variable? Does it still exist? What happens when we print(name). Will it show the original value “Kalu Kalu” or the new “Kalu Times 2”. Let’s find out.

```
>>> print(name)
Kalu Kalu
```

As you can see, the original value of name has not been changed. This teaches an important lesson; when we did nickname = name , this created an ENTIRELY NEW OBJECT, not merely a reference to our original name object. Changing the value of nickname will not affect name at all because name and nickname are 2 completely different objects. Even though name was used as a reference to create nickname, changing nickname afterwards WILL NOT AFFECT name. Because a variable assignment creates a completely different copy, not merely a reference to the original object.

Statements & Operations

Statements

Statements are a very important part of programming. Just like in Math, a statement is evaluated to it's simplest form.

From here on out we're going to run our python code by saving it in a script file and running it in our Bash shell.

First, open your Terminal and in your Desktop create a folder called python_book. Inside that folder we're going to create a file called ex1.py and use Sublime to write our code inside of it.

```
Kalus-MacBook-Pro:~ kalukalu$ cd Documents/
Kalus-MacBook-Pro:Documents kalukalu$ mkdir python_book
Kalus-MacBook-Pro:Documents kalukalu$ cd python_book/
Kalus-MacBook-Pro:python_book kalukalu$ touch ex1.py
```

Pic16

Click on the spotlight search icon on the top right corner of your screen and search and open Sublime Text.

Pic 17

Once Sublime Text is open Click on File > Open (if you are on Ubuntu click on Open Folder).

Pic18

The find and open your python_book folder.

Pic19

Lastly click and open ex1.py.

20

A Simple Expression

Inside of ex1.py please type the following code.

A note on Comments:

You do not have to type any lines of code that start with a #. Any code followed by the # sign is a comment and will be ignored by the complier at runtime. It Is merely explanatory to help explain a code segment.

ex1.py

```
# You go to the mall and you purchase
# one item for two dollars and forty cents.
#  and another for five dollars and six cents.
# What is your total bill?
total = 2.40 + 5.06
print("Here is the total bill")
print(total)
```

pic21

Don't forget to save the file. Then open your Terminal and use your python command to run your ex1.py program. Your output should look like this.

```
Kalus-MacBook-Pro:python_book kalukalu$ python ex1.py
Here is the total bill
7.46
```

Pic22

Expression Order of Operations: PEMDAS

What happened? First the expression on the right side of the equals sign (2.40 + 5.06) is evaluated. Generally, just like in math, the order of operations in which expressions are evaluated follow the PEMDAS rule, standing for, Parenthesis, Exponents, Multiplication, Division, Addition, Subtraction. After the expression is evaluated it is then assigned to the variable to the left of the equal sign.

A More Complex Expression

Let's try a more complex express. 2 + 2 * 3 . Before we type the code, can you guess what this Math expression will evaluate to following the order of operations rules you learn in grade school? Think about it for a second and then create another file called ex2.py . Type the following code inside of your new ex2.py file.

ex2.py

```
# Pens And Pencils both cost $2.
# You buy three pens and one pencil.
# What is your total?
total = 2 + 2 * 3
print(total)
```

Pic23

After you have typed and saved it, open your terminal. In case you closed your past terminal cd back into your python_book folder so that we can run the newly created ex2.py script that is in there. Then run your script.

```
Kalus-MacBook-Pro:python_book kalukalu$ cd ~/Documents/python_book/
Kalus-MacBook-Pro:python_book kalukalu$ python ex2.py
8
```

Your response should be six, the same answer that you should have received when doing it by hand. According to order of operations, when we have the expression 2 + 2 * 3 , we first evaluate the multiplication because multiplication (M) comes before addition (A) in order of operations (PEMDAS). Therefore we first must evaluate the statement 3 * 2. The expression 3 * 2 evaluates to 5. Now we are left with the expression 2 + 6 , which equals 8. Therefore the value 8 is assigned to the variable name total.

Expressions & Variables

Expressions work the exact same way with variables. In ex3 we're going to recreate what we did in ex2 except with variable names instead. Please create a new file called ex3.py.

A Note On Exercise Names:

You will notice a pattern. Every new exercise will have a new file with a new filename that you will have to create. The filenames will be named by adding one to the last exercise number, for example, ex4.py , ex5.py ex6.py and so on.

Please make sure that you take note of which exercise we're in.

ex3.py

```
# Pens And Pencils both cost $2.
# You buy three pens and one pencil.
# What is your total?
pen = 2
pencil = 2
total = pen + pencil * 3
print(total)
```

As you can probably already guess, when we run this in our terminal we will get the same output as with ex2.py.

```
Kalus-MacBook-Pro:python_book kalukalu$ python ex2.py
8
Kalus-MacBook-Pro:python_book kalukalu$ python ex3.py
8
```

Order of operations work the same whether you are using the actual number, or a variable name as a placeholder.

Expressions & Parenthesis

Just like in math, you can use parenthesis for order of operations reasons, or just to make your code more clear.

For instance, try this exercise:

ex4.py

```
# Pens And Pencils both cost $2.
# You buy three pens and one pencil.
# What is your total?
pen = 2
pencil = 2
total = pen + pencil * 3
correct_order = pen + (pencil * 3)
incorrect_order = (pen + pencil) * 3
print(total)
print("This is the correct order of operations")
print(correct_order)
print("This is an  order of operations of the original question")
print(incorrect_order)
```

When you run the code, this should be your output:

```
Kalus-MacBook-Pro:python_book kalukalu$ python ex4.py
This is the default order of operations
8
This is the correct order of operations
8
This is an  order of operations of the original question
12
```

Operators

You have already been exposed to many operators so far in this book. For instance, the addition and multiplication operator. When we did 2 + 3 * 2 .

You have also been introduced to probably the most vital operator, the assignment operator, = . When we wrote total = 2 + 3 * 2 , the assignment operator acted to assign the result of the expression 2 + 3 * 2 to the variable name total. As we stated earlier, the expression to the right of the assignment operator is first evaluated, and then the result is assigned to the variable name to the right of the operator.

Pic24

Here is a list of basic math operators:

1. Addition
2. Multiplication.
3. Subtraction

4. Division
5. Modulus
6. Power

There are your basic Math operators. But you also have other types of operators such as comparison operators. Here the list of comparison operators

1. Greater than >
2. Less than <
3. Greater than or >=
4. Less than or equal <=
5. Equal ==

Before we test out the comparison operators, it is worth nothing, do not get the double equals equality comparison operator, == , confused with the single equals assignment operator mentioned earlier, = . They are not they same and confusing is the cause of errors for beginner programmers and even experienced absent minded programmers (myself included). As we will discuss more below, the equality comparison checks to see if the value on the left of the == is equal to the value to the right of it. Equality comparisons always equate to a True or False Boolean expression. For example 2 + 3 == 6 -1 would evaluate to the expression True.

Order of Operations

Some text

Python Data Types

Some text

1) Strings

Creating Strings

Substring Selection

2) Ints

Integers

Floats

3) Floats

Creating Floats

Common Methods

4) Lists

Creating Lists

List Item Selection

5) Tuples

Creating Tuples

Substring Selection

6) Dictionaries

Creating Dictionaries

Querying a Dictionary

7) Booleans

Creating Floats

SubString Selection

Control Structures & For-Loops

Some text

If Statements

Some Text

Functions

Some Text

Classes

Some Text