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.

A note on typing code: You should only type the part that comes after >>> . Do not type the number or the >>> . So for instance, for the first line of code, 1. >>> bill = 54.87 , you should only type bill = 54.87 in your terminal.

Please type the following:

```
1. >>> bill = 54.87
2. >>> tip_percent = 0.10
3. >>> tip = bill * tip_percent
4. >>> total = tip + bill
5. >>> print("Here is your total tip amount")
6. >>> print(tip)
7. >>> print("Here is your total bill")
8. >>> 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 (pronounced dot pie) 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:

A note on typing code: You should only type the part that comes after the number . Do not type the number. So for instance, for the first line of code, bill = 54.87 , you should only type bill = 54.87 in your sublime calculate_bill.py file.

calculate_bill.py

```
1. bill = 54.87
2. tip_percent = 0.10
3. tip = bill * tip_percent
4. total = tip + bill
5. print("Here is your total tip amount")
6. print(tip)
7. print("Here is your total bill")
8. 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 will mostly type our python code in a script file, save that script file and then run that script file from our Bash shell. But, whenever we want to quickly test out code we can use the python shell instead.

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 app.py. Please make sure that you create these two files, app.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 app.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

```
1. name1 = 'Jon'
2. name2 = 'Barry'
3. name3 = 'Eke'
4. name4 = 'Uche'
5.
6. age1 = 24
7. age2 = 45
8. age3 = 62
9. age4 = 14
```

Import Objects One At A Time

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

app.py

```
1. from data import name1, age1
2.
3. print(name1)
4. 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 app.py python.

```
Kalus-MacBook-Pro:~ kalukalu$ cd ~/Desktop/
Kalus-MacBook-Pro:Desktop kalukalu$ python app.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 app.py. Great! But what if we wanted to print name2 or age2? Well, for every object we wanted to use inside of app.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.

A note on comments: If you see some a number with a grey background, `1` , do not type it. That is just a comment. So below, when you see 1. from data import * `1` , only type, *from data import **` .

app.py

```
1. from data import * 1
2.
3. print(name1, name2, name3, name4, age1, age2, age3, age4)
```

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

```
Kalus-MacBook-Pro:Desktop kalukalu$ python app.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 1 . 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 app.py to look like the following:

app.py

```
1. import data
2.
3.
4. 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 app.py
```

Your output should like this:

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

When you use this method to import, do not forget that always use the module name followed by a dot (i.e, data.name3) to try to access any of that modules

The if __name__ == “__main__” Use-Case

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

apy.py

```
1. from calculate_bill import bill
2.
3.
4. print(bill)
```

And run it in your Terminal.

```
Kalus-MacBook-Pro:Desktop kalukalu$ python app.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 app.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 app.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

```
1. bill = 54.87
2. tip_percent = 0.10
3. tip = bill * tip_percent
4. total = tip + bill
5.
6. if __name__ == "__main__":
7.     print("Here is your total tip amount")
8.     print(tip)
9.     print("Here is your total bill")
```

10. print(total)

And now let's run app.py again.

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

Now only the information we wanted printed in app.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

Variable Name Errors:

Variable Names Are Case Sensitive

A common error in programming is what's known as the NameError. Computers interpret language differently than humans. If you wrote written instructions to your secretary that slogan = "To make the world a better place", and then asked him to print slogan, he could understand to print "To make the world a better place". He could assume your intention even though slogan is initially lowercase when you defined the variable, but you told him to print Slogan using an upper case S. However, a computer has no such intuitive skills. Instructions to a computer need to be precise or it will be confused. That is because in programming variable names are case-sensitive. To a computer, slogan and Slogan are not the same objects. They are two completely different objects, one that has been defined as meaning "To make the world a better place" and another than has not been defined at all. If you told a computer to print Slogan instead of slogan, it would tell you, "NameError: name 'name' is not defined".

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 its 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.

If You Have Mac/Linux:

```
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  1
```

If You Have Windows:

```
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$ type nul > ex1.py  1
```

The touch command: Mac/Linux: If you have Mac or linux the touch command `1` creates a new file. The touch command takes an argument, the name of the file you want to create. Here we tell it to create a file named ex1.py . Windows does not have the touch command, you must instead use a different command. See below if you are on windows.

The touch command: Mac/Linux: If you have Windows the type nul > ex1.py command `1` creates a new file called ex1.py. Windows does not have the touch command, you must instead use a different command.

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 # Signs:

You do not have to type any lines of code 1 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

```
1. # You go to the mall and you purchase 1
2. # one item for two dollars and forty cents. 1
3. #  and another for five dollars and six cents. 1
4. # What is your total bill? 1
5.
6. total = 2.40 + 5.06
7. print("Here is the total bill")
8. 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.

```
Kalus-MacBook-Pro:python_book kalukalu$ python ex1.py
```

Your output should look like this.

```
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
```

Your response should be 8, the same answer that you should have received when doing it by hand.

8

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.py we're going to recreate what we did in ex2.py 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

```
1.  # Pens And Pencils both cost $2.
```

```
2. # You buy three pens and one pencil.
3. # What is your total?
4. pen = 2
5. pencil = 2
6. total = pen + pencil * 3
7. 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

```
1. # Pens And Pencils both cost $2.
2. # You buy three pens and one pencil.
3. # What is your total?
4. pen = 2
5. pencil = 2
6. total = pen + pencil * 3
7. correct_order = pen + (pencil * 3)
8. incorrect_order = (pen + pencil) * 3
9. print(total)
10. print("This is the correct order of operations")
11. print(correct_order)
12. print("This is an  order of operations of the original question")
13. 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

Math Operators

Here is a list of basic math operators:

1. Addition
2. Multiplication.
3. Subtraction
4. Division
5. Modulus
6. Power

Comparison/Equality Operators

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 ==

Do Not Confuse the Equality (==) Operator With the Assignment Operator (=)

Before we test out the comparison operators, it is worth noting, 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).

Comparison Operators Return True or False Boolean Values

As we will discuss more below, comparison operators (for example, ==) 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.

Comparison/Equality Operators

Operators Practice

The main thing to know for order of evaluations when it comes to operators is that they are pretty intuitive. Arithmetic operators are evaluated first, then the others.

Before we get started let's discuss the modulus operator, %. The % operator gives yoo the remainder when dividing by a number. For example, 3 % 2 will return 1 because 3 / 2 leaves a remainder of one. Ok, now let's get started. Before you type and run the code try to guess what the output will be. After you guess go ahead and type and run the following code.

ex5.py

```
1. remainder = 5 % 2
2. boolean1 = remainder == 1
3. boolean2 = remainder >= 2
4. boolean3 = boolean2 and boolean1
5. boolean4 = boolean2 or boolean1
6. boolean5 = True and False or True and True or False and True or False or True
7. num1 = 2 + 3 ** 2 - (3 + 3)
8. num2 = num1  + 19 % 17 * 4 - 1
9. num3 = num1 / 2 + num2
10.
11. print(remainder)
```

```
12. print(boolean1)
13. print(boolean2)
14. print(boolean3)
15. print(boolean4)
16. print(num1)
17. print(num2)
18. print(num3)
```

Did you get the expected you guess? Can you create your own fun brain teasers?

Python Data Types

The type() Function

Before we get started learning about python data types we first have to learn …….

1) Strings

How to Create A String

Strings are a fundamental data type that we have already been working with. A string is created by merely wrapping quotes, single or double, around any text. Let's create several example strings.

ex6.py

```
1.  string1 = 'How are you doing'
2.
3.  string2 = "You can create a string this way as well."
4.  string3 = "Strings can also have numbers, like 23 414, and some symbols, like #$*&#"
5.  intro1 = "Strings are used to display text that is meant for human eyes."
6.  print("\n\nHere is the first intro 1")
7.  print(intro1)
8.  intro2 = " You can add one string to another string. "
9.
10. # You can use intro1 in the assignment even though you are overriding it
11. intro1 = intro1 + intro2
12. # IT's very comment to see the same variable name on both sides
13. # of the equals sign like above. It's okay
```

```
14. # as long as that variable has already been created before
15.
16. print("\n\nHere is string 1")
17. print(string1)
18. print("\n\n Here is string 2")
19. print(string2)
20. print("\n\n Here is string 3")
21. print(string3)
22. print("\n\n Here is the second intro 1")
23. print(intro1)
24. print("\n\n Here is intro2")
25. print(intro2)
```

Did you notice that strings had a funny \n ini them. Some characters inside of a string definition have special meaning, we call these characters escape characters. If you do \n the will tell python to add a new line when displaying this string for humans to see.

Substring Selection

How do you select one single character from a string? How do you select a substring of characters from a string? That is the topic of substring selection, also known as string slicing.

If you have a string, quote = “We have a dream.” , how do you select a piece of that string? The answer is string select with “bracket-notation”. Bracket refers to [ ] . To select a sub-string you just add [ ] to the string and write the start and stop points you want. name[0:3] would give me the first to characters 0, 1, and 2, or “I h”

Please run the following code.

ex7.py

```
1. quote = "We have a dream"
2. substring1 = quote[1]
3. substring2 = quote[-1]
4. substring3 = quote[0:3]
5. substring4 = quote[1:4]
6. substring5 = quote[:]
7. substring6 = quote[:7]
8. substring7 = quote[6:-1]
9. substring8 = quote[-5:]
10.
11. print(substring1)
12. print(substring2)
13. print(substring3)
```

```
14. print(substring4)
15. print(substring5)
16. print(substring6)
17. print(substring7)
18. print(substring8)
```

Can you guess what the output should be? Try to guess first before looking. The answer is below:

```
e
m
We h
e h
We have a dream
We have
e a drea
dream
```

Position Numbers Start With 0

As you may have just noticed, when discussing position numbers in programming we usually start with zero. For instance, the first character in the string “We have a dream” is the “W”. The number we used use to describe this position is position 0, not position 1. The second character, “e” , would position 1, and so on.

When Doing String Selection With Bracket Notation The Second Argument Is Exclusive

From Position X Up Until Position Y

Earlier we had a piece of code that said substring3 = quote[0:4]. Surprising this printed our “We h” instead of “We ” . That is because the second number is EXCLUSIVE. That means we do not include the character at that position, we include everything UP UNTIL. On the other hand, the first number is always include. So you should think of [0:4] as saying give me all the characters **from** position 0 **up until** position 4.

The Str() Method

In the next method we will learn about number types like Integer and Float. Sometimes you may want to concatenate a number with a string. Doing so will give you a error that reads *TypeError: cannot concatenate ‘str’ and ‘int’ objects*. You must first convert the int to a string using the str() function. Please type the script below.

ex8.py

```
1. apples = 10
2.
3. text = "I have " + apples + " apples.”   1
4. print(text)
5. print("Original apples datatype -->", type(apples))
```

If you run this you will get an error that looks like this.

```
Traceback (most recent call last):
  File "ex8.py", line 3, in <module>   2
    text = "I have " + apples + " apples.”   3
TypeError: cannot concatenate 'str' and 'int' objects   4
```

This gives us a good opportunity learn how to read Traceback calls. In python, whenever your code has an error that stops your program from running, you will receive an a Traceback call and a hopefully descriptive message about the issue. Resist the urge to freak out at the error message. It usually can be deciphered quite easily with just a little bit of careful reading.

Let's read the first line of our Traceback call **2** . It tell us the file and line number that our error is in. This is very helpful. We now know that the bug is in ex8.py, line 3, **1** . The Traceback call is even helpful enough to print out the specific line of code for us **3** .Finally, the last line of Traceback specifically prints out our error message , **4** . From this error message we can see that the error is exactly what I said it would be; *TypeError: cannot concatenate 'str' and 'int' objects.*

The reason for this error is that you cannot add a string object with an integer object. If you wish to concatenate a string and an integer, you must first convert the integer into a string using the str() function. Please write the following script:

ex9.py

```
1. apples = 10
2.
3. text = "I have " + str(apples) + " apples.”   1
4. print(text)
5. print("Original apples datatype -->", type(apples))   2
6. print("apples datatype After str() -->", type(str(apples)))   3
```

Our code should now work now.

```
I have 10 apples.
('Original apples Datatype -->', <type 'int'>)   4
('apples Datatype After str() -->', <type 'str'>)   5
```

The difference is in the with the str() function **1** that we wrapped around our apples object. The string function returns the string version of the number you give it. Since we used the type() function to print out the original apple object's data type **2** , we can see that our original apples object is an int type **4** . But after we convert it **3** , we can now see that it is now indeed a str **5** therefore avoiding the TypeError.

String Quiz

ex10.py

```
1. quote = "Practice makes perfect"
2.
3. substring1 = quote[0:8]
4. substring2 = quote[-7:]
5. substring3 = quote[9:14]
6. substring4 = substring2 + " " + substring1 + " " + substring3
7. substring5 = substring3 + " " + substring2 + " " + substring1
8. substring6 = substring1 + " " + substring3 + " " + substring2
9. substring7 = substring3 + " " + substring1 + " " + substring2
10.
11. story = "I once heard that "
12. story += substring4 + " "
13. story += " the saying " + substring6 + " true. "
14. story += "So we must " + substring1 + " " + substring2
15. story += " to be " + substring2
16. story += "."
17.
18. print(substring1)
19. print(substring2)
20. print(substring3)
21. print(substring4)
22. print(substring5)
23. print(substring6)
24. print(substring7)
25. print(story)
```

QUESTION 1:

What will be the output of print(substring1)?

QUESTION 2:

What will be the output of print(substring2)?

QUESTION 3:

What will be the output of print(substring3)?

QUESTION 4:

What will be the output of print(substring4)?

QUESTION 5:

What will be the output of print(substring5)?

QUESTION 6:

What will be the output of print(substring6)?

QUESTION 7:

What will be the output of print(substring7)?

QUESTION 8:

What will be the output of print(story)?

---

2) Ints

Integers are created by just typing a number without any quotes surrounding it. For instance age = 10 will create an Integer object with the value 10. Integers can only be whole numbers.

Integer Division Can Have Funny Results

Let's test something out real quick. Type up the following result. Before you run the code, do the math by hand. Then run the code and see if you got the expected result.

ex11.py

```
1. assists = 5
2. turnovers = 3
3. ratio = assists / turnovers
4.
5. print(ratio)
```

Surprised by the result? When we run this code our terminal returns 1 instead of 1.5.

```
1
```

The reason we get 1 instead of 1.5 is because when we do integer division, all the decimal information gets truncated (cut off).

Always Divide With At Least One Float To Avoid Truncation

In order to get a real number (a number with decimal points) as a result, you must divide using at least one real number. As we will learn in the following section, python represents real numbers using the float data type. Please type the following code.

ex12.py

```
assists = 5.0
turnovers = 3
```

```
ratio = assists / turnovers

print(ratio)
```

Now our result is 1.5 as it should be.

```
1.5
```

The magic was changing 5 to 5.0. As long as there is a decimal point in a number that number will be a float type. We could have changed turnovers to a float as well. It doesn't matter, as long as at least one number in the arithmetic expression is a float then the expression will be evaluated as a float.

You Can Use The float() Function to Convert an Int into a String

We converted the int into a string by use what's called a "floating point *literal.*" Literal means that means that we type characters in a certain syntax and python will automatically convert the characters into the corresponding data type. For instance, we already have seen string literals and integer literals. if you type characters enclosed using quotation marks that python will create a string of those characters. If you type just numbers without any quotation marks python will create an integer of that number. object creates a string. We used a float literal above by typing 5.0.

But sometimes we do not have access to the actual place where we created the variable. We only have a variable name that we want to convert. In such cases we can use the float() function. The code above works just as well as the code below.

ex13.py

```
1. assists = 5
2. turnovers = 3
3. ratio = assists / float(turnovers)
4.
5. print(ratio)
```

Just like before our result is 1.5.

```
1.5
```

All we did was wrap float() around turnovers, float(turnovers), to convert it into a float before we divided.

ex14.py

```
1. num1 = 5 / 3.0 + 2 * 3
2. num2 = 5 / 3 + 2.0 * 3
3.
4. print(num1)
```

5. print(num2)

The result should be

```
7.66666666667
7.0
```

The Expected Result: num1
What happened was that in num1 the expression 5 / 3.0 was first evaluated. Since that expression included a float it was evaluated as 1.67. then 2 * 3 was evaluated as 6. 6 + 1.5 of course equals 7.67, the expected result.

The Unexpected Truncated Result: num2
However in num2 we got a wrong result. This is because the expression 5 / 3 was first evaluated, resulting in 1. Because this expression only included integers the .67 was truncated. Next 2.0 * 3 was evaluated. Because 2.0 is a float, the result of the expression, 6.0, is also a float. Finally we add 6.0 with 1 and get the float 7.0 as a result.

The takeaway is that to get the correct result with integer division It is not enough to just have a float somewhere in the expression unrelated to the actual division. You must convert one of the floats in the immediate division expression itself.

3) Floats

We just learned that whole numbers are represented by the Int data type in Python. But what if we want a real number, a number with a decimal point? Real numbers are represented by the Float data type. Here is an example of how a float is created.

ex15.py

1. gpa = 2.0
2. print(gpa)

Converting A Float to an Int

We learned in the int section that we can convert a int type to a float type. We can conversely convert a float type to an int type.

ex16.py

1. students = 20.0
2.
3. print("\n\nStudents is currently a float type")
4. print(type(students))
5. print(students)
6.
7. students = int(students)

```
8.
9.  print("\n\nNow students has been converted into an int type")
10. print(type(students))
11. print(students)
```

The output should look like this.

```
Students is currently a float type
<type 'float'>
20.0


Now students has been converted into an int type
<type 'int'>
20
```

4) Lists

Lists are a collection of objects. For instance of you can have a list of integers, a list of floats, a list of strings, or a list of any combination of the above. You can even have a list of lists (what's known as a two dimensional list (or in Mathematics a matrix). Let's create some lists.

To create a list we simply start with opening and closing brackets, [ ] . Inside the opening and closing brackets we insert any python objects that we want, separated by a comma. Enough talking, let's create some lists!

ex17.py

```
exam_grades = [80, 93, 85, 94]
students = ['eke', 'jordan', 'uche', 'nnenna']
gpas = [3.8, 3.6, 3.33, 4.0]

print(exam_grades)
print(students)
print(gpas)
```

Congrats on creating a list. Now let's do some simple list selection to see how to select elements from the list.

List Item Selection

Now that we have created let list let's see how we can select items from a list. Selecting items from a list is similar to substring selection that we discussed earlier. To select items from a list you use bracket-notation.

ex18.py

```
exam_grades = [80, 93, 85, 94]
students = ['Eke', 'Jordan', 'Uche', 'Nnenna']
gpas = [3.8, 3.6, 3.33, 4.0]

uche_grade = exam_grades[2]
uche_name = students[2]
uche_gpa = gpas[2]

first_two_grades = exam_grades[0:2]
last_two_grades = exam_grades[2:4]

print(uche_grade)
print(uche_name)
print(uche_gpa)
print(first_two_grades)
print(last_two_grades)
```

Our output should look like this:

```
85
Uche
3.33
[80, 93]
[85, 94]
```

Let's discuss a bit of what happened. Just like with string substring we can select items with bracket-notation. Bracket refers to [ ] . To select an element from the list you just add [ ] to the list name and write the start and stop points you want.

Selecting a Single Item

If you have a list, students = ['eke', 'jordan', 'uche', 'nnenna'] , and we want to select the third name in the list, we would do student[2]. Do not forget that in programming we often start counting with 0. Since students[0] would give us the first position, students[2] is actually the third position number.

Selecting Multiple Items

How do you select multiple items from a list? We select items from a list with "bracket-notation" and adding two position numbers separated by a colon . For instance, students[0:2]

would give me the first two items at positions 0 through 1, ['eke', ‘jordan’]. As we will discussed with strings selection, and will discuss again below, when selecting using bracket-notation the second number after the colon is always **EXCLUSIVE.** Python interprets the statement students[0:2] as meaning get me items from positions 0 **up until but not including** position 2.

When Doing String Selection With Bracket Notation The Second Argument Is Exclusive

From Position X Up Until Position Y

Earlier we had a piece of code that said students[0:2] Surprising this gave us ['eke', ‘jordan’] and not ['eke', 'jordan', ‘uche']. That is because the second number is EXCLUSIVE. That means we do not include the element at that position, we include everything UP UNTIL. On the other hand, the first number is always include. So you should think of [0:2] as saying give me all the objects **from** position 0 **up until** position 2.

Let’s use our lists to draft a little story about each student.

Position Numbers Start With 0

As you may have just noticed, when discussing position numbers in programming we usually start with zero. For instance, the first element in the students list, students = ['Eke', 'Jordan', 'Uche', ‘Nnenna’], is “eke”. The number we used use to describe this position is position 0, not position 1. The second element, “jordan” , would position 1, and so on.

When Doing String Selection With Bracket Notation The Second Argument Is Exclusive

From Position X Up Until Position Y

Earlier we had a piece of code that said students[0:2] Surprising this gave us ['eke', ‘jordan’] and not ['eke', 'jordan', ‘uche']. That is because the second number is EXCLUSIVE. That means we do not include the element at that position, we include everything UP UNTIL. On the other hand, the first number is always include. So you should think of [0:2] as saying give me all the objects **from** position 0 **up until** position 2.

Let’s use our lists to draft a little story about each student.

ex19.py

```
1. exam_grades = [80, 93, 85, 94]
```

```
2. students = ['Eke', 'Jordan', 'Uche', 'Nnenna']
3. gpas = [3.8, 3.6, 3.33, 4.0]
4.
5. text1 = "\n\nHi "
6. text2 = ". This is Professor Obama. Congratulations on finishing your
   Constitutional Law exam. "
7. text3 = "Your final grade was a "
8. text4 = ". Now your gpa is a "
9.
10. eke_story = text1 + students[0] + text2 + text3
11. eke_story += str(exam_grades[0]) + text4 + str(gpas[0])
12.
13. jordan_story = text1 + students[1] + text2 + text3
14. jordan_story += str(exam_grades[1]) + text4 + str(gpas[1])
15.
16. uche_story = text1 + students[2] + text2 + text3
17. uche_story += str(exam_grades[2]) + text4 + str(gpas[2])
18.
19. nnenna_story = text1 + students[3] + text2 + text3
20. nnenna_story += str(exam_grades[3]) + text4 + str(gpas[3])
21.
22.
23. print(eke_story)
24. print(jordan_story)
25. print(uche_story)
26. print(nnenna_story)
```

Our output will look like this:

```
Hi Eke. This is Professor Obama. Congratulations on finishing your
Constitutional Law exam. Your final grade was a 80. Now your gpa is a
3.8


Hi Jordan. This is Professor Obama. Congratulations on finishing your
Constitutional Law exam. Your final grade was a 93. Now your gpa is a
3.6


Hi Uche. This is Professor Obama. Congratulations on finishing your
Constitutional Law exam. Your final grade was a 85. Now your gpa is a
3.33
```

```
Hi Nnenna. This is Professor Obama. Congratulations on finishing your
Constitutional Law exam. Your final grade was a 94. Now your gpa is a
4.0
```

Do not be confused with str(exam_grades[2]) . Let's break down what this means. This means we grab the element in position two of exam_grades. This is element the integer, 85 (remember for position numbers we start counting at 0). In order to add a integer to a string we must first convert it to a string. str(exam_grades[2]) will convert the integer 85 into a string, '85', that way we can do string concatenation in order to make our story. First the expression exam_grade[2] is evaluated. This is evaluated into the element 85. After str(85) is evaluated, resulting in "85". Then we add

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

Classes

Class Methods

Built-In Methods

Interacting With Your Operating System: The os library

Algorithm Practice

Exam

What You Now Know

What You Still Don’t Know