**CS469 Data Structures and Algorithms**

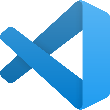
**HOS00C – Python Basics**

03/29/2023 Updated by Christopher Sharp

01/04/2024 Reviewed by Anh Nguyen

09/26/2024 Reviewed by Shahid Khan

School of Technology and Computing (STC) @City University of Seattle (CityU)

 A picture containing clock

Description automatically generated

**Before You Start**

* The Python HOS assignments (HOS00C and HOS00D) are OPTIONAL. If you already learned Python in other courses, feel free to skim the document.
* The directory path shown in screenshots may be different from yours.
* Some steps are not explained in the tutorial**.**If you are not sure what to do:

1. Consult the resources listed below.
2. If you cannot solve the problem after a few tries, email STC TA Center for help. (stctacenter@cityu.edu)

**Learning Outcomes**

Students will be able to:

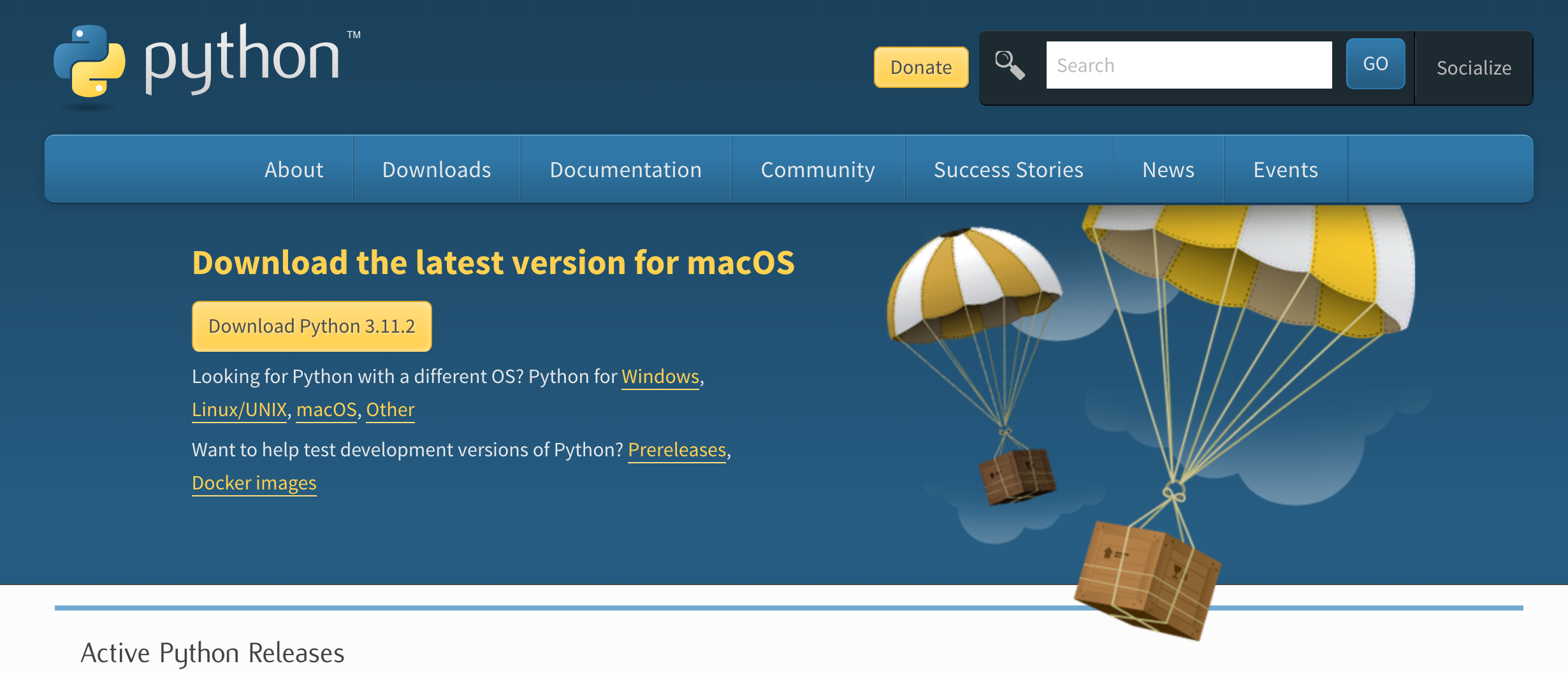
* Set up the working environment for Python
* Understand the basics python concepts
* Understand the data conversion in Python

**Install Python in your machine**

We will use Python 3 instead of Python 2 for all demonstrations. If you have already installed Python3, please skip this section.

1. Click the following link to visit Python downloading page <https://www.python.org/downloads/>

1. Click the downloading link which should detect the operating system you are using:



Download the latest version (Python 3.11.2 above)

1. Perform the installation process:

Click on the available version in the yellow button. After downloading the files, follow the instructions on your screen to install python.

1. After the installation, in the VS code terminal, type the following command to test it:

>>> **python3 --version**

You should see whatever version you installed comes up: “3.11.2” in the above case.

1. The python extension is also recommended to install in the VS Code.

Click the **View** -> **Extension** in the VS Code menu

A screenshot of text

Description automatically generated

Search python and click the install button on the right

A screenshot of a cell phone

Description automatically generated

**Python Basic**

**Variables**

Variables are containers for storing data values. Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume). Rules for Python variables:

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive (age, Age and AGE are three different variables)

**Variable creation**

1. Below is an example of how variables are created. You don’t have to type the code. It is just for your understanding.

The python code is typed under a python file with “.py” extension. e.g. **filename.py**

**>>>**m**essage = “Hello World”**

The “message” is the variable. Variables do not need to be declared with any type and can even change type after they have been set.

To display the contents inside the message use ‘print’ function.

>>> **print(message)**

To compile and run the python file type the below in the terminal

>>> **python3 filename.py**

The above code will display the following output.



1. Look at the example below to understand how variables work. You don’t have to type the code. It is just for your understanding.

A screenshot of a cell phone

Description automatically generated

When the ‘message’ was used the first time its data type was string. During the second time the data type got changed to integer as the value of the ‘message’ is assigned to 5.

The variable ‘message’ has been used twice and the recently assigned value to the variable ‘message’ is its final value. (i.e.) the value ‘Hello World’ has been replaced with 5.

You can change the value of a variable in your program at any time, and Python will always keep track of its current value.

The output of the above code is

A picture containing food

Description automatically generated

**Multiple assignments**

Python allows you to assign values to multiple variables in one line andyou can assign the same value to multiple variables in one line

1. Create a file by name **variable.py** and type the following code.

A close up of text on a black background

Description automatically generated

1. Check the output of the above code in the VS code terminal.

>>> **python3 variables.py**

A close up of a logo

Description automatically generated

**Data Types**

Data types are the classification or categorization of data items. Data types represent a kind of value which determines what operations can be performed on that data. Numeric, non-numeric and Boolean (true/false) data are the most used data types.

**Numeric**

A numeric value is any representation of data which has a numeric value. Python identifies three types of numbers:

* **Integer:** Positive or negative whole numbers (without a fractional part). For example:

>>> x = 5

* **Float:** Any real number with a floating-point representation in which a fractional component is denoted by a decimal symbol or scientific notation. For example:

>>> y = 2.5

* **Complex number:** A number with a real and imaginary component represented as x+yj. x and y are floats and j is -1(square root of -1 called an imaginary number). For example:

>>> z = 2 + 3i

**Boolean**

Data with one of two built-in values True or False. Notice that 'T' and 'F' are capitalized. The true and false are not valid Booleans and Python will throw an error for them. For example:

>>> **print (10>9)**

>>> True

**Sequence Type**

A sequence is an ordered collection of similar or different data types. Python has the following built-in sequence data types:

* **String**: A string value is a collection of one or more characters put in single, double or triple quotes. The string is initialized in the following ways.

>>> message = “This is a string declaration”

>>> message = ‘This is also a string declaration’

* **List**: A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.

>>> list1 = ['physics', 'chemistry', 1997, 2000];

* **Tuple**: A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses. Tuple is not mutable i.e. it doesn't have any methods for changing its contents).

>>> **tup1 = ('physics', 'chemistry', 1997, 2000)**

**Dictionary**

A dictionary object is an unordered collection of data in a key: value pair form. A collection of such pairs is enclosed in curly brackets. For example:

 >>> **dic1 = {1:"Steve", 2:"Bill", 3:"Ram", 4: "Farha"}**

**String functions**

Python has a set of built-in methods that you can use on strings. A few examples are given below.

1) Create another file **Strings.py** and type the following code.

A picture containing cup, drawing, food

Description automatically generated

A picture containing drawing

Description automatically generated

2) Type the following to get the output of the code in the terminal

>>> **python3 Strings.py**



**Explanation:**

The title() method returns a string where the first character in every word is upper case. If the word contains a number or a symbol, the first letter after that will be converted to upper case.

The upper() method returns a string where all characters are in upper case. Symbols and Numbers are ignored.

The lower() method returns a string where all characters are lower case. Symbols and Numbers are ignored.

**Concatenation:**

In Python, there are a few ways to concatenate – or combine - strings. The new string created is called a string object. Obviously, this is because everything in Python is an object – which is why Python is an object-oriented language.

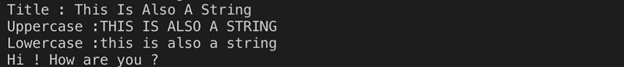
3) In the same **Strings.py**, add the code below.

A close up of a screen

Description automatically generated

4) Type the following to get the output of the code in the terminal

>>> **python3 Strings.py**



**Explanation:**

Also called “formatted string literals,” f-strings are string literals that have an f at the beginning and curly braces containing expressions that will be replaced with their values. The expressions are evaluated at runtime and then formatted using the \_\_format\_\_ protocol.

**Numbers and operators**

Python language supports the following types of operators.

* Arithmetic Operators
* Comparison (Relational) Operators
* Assignment Operators
* Logical Operators
* Bitwise Operators
* Membership Operators
* Identity Operators

The commonly used operators are discussed below.

**Arithmetic operator:**

Consider value of a = 10, b = 20

A screenshot of a cell phone

Description automatically generated

1) Create a file **Numbers.py** and type the following code.

A screenshot of a cell phone

Description automatically generated

2) Type the following to get the output of the code in the terminal

>>> **python3 Numbers.py**

A close up of a logo

Description automatically generated

If you look at the answer of a+c and a\*\*b, it is displayed as an arbitrary number of decimal places. This happens in all languages and is of little concern. Python tries to find a way to represent the result as precisely as possible, which is sometimes difficult given how computers must represent numbers internally.

**Comparison Operator:**

Consider value of a = 10, b = 20

A screenshot of a cell phone

Description automatically generated

**Logical Operator:**

Consider value of a = 10, b = 20

A screenshot of a cell phone

Description automatically generated

**Membership Operator:**

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples.

A screenshot of a cell phone

Description automatically generated

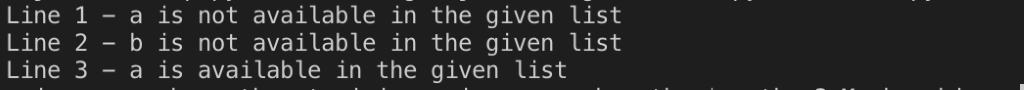
1) Create a file **Membership.py** and type the following code.

A screenshot of a cell phone

Description automatically generated

2) Type the following to get the output of the code in the terminal

>>> **python3 Membership.py**



**Identity Operator:**

Identity operators compare the memory locations of two objects.

A screenshot of a cell phone

Description automatically generated

1) Create a file **Identity.py** and type the following code.

A screenshot of a cell phone

Description automatically generated

2) Type the following to get the output of the code in the terminal

>>> **python3 Identity.py**

A picture containing bottle, black, sign, red

Description automatically generated

**User Input**

To receive information through the keyboard, Python uses the input() function. This function has an optional parameter, commonly known as prompt, which is a string that will be printed on the screen whenever the function is called.

When the input() functions is called, the program flow stops until the user enters the input via the command line. To enter the data, the user needs to press the ENTER key after inputting their string. While hitting the ENTER key usually inserts a newline character ("\n"), it does not in this case. The entered string will simply be submitted to the application.

The input() function, by default, will convert all the information it receives into a string.

1. Create a new file **Userinput.py** and type the following code

A close up of a screen

Description automatically generated

1. Type the following in the terminal to check the output of the above code

>>>**python3 Userinput.py**

A picture containing drawing

Description automatically generated

If you look at the type of variable “age”, it is displayed as str which is string. But we mentioned numeric value for age. Now let’s change the way we get input from the user.

1. In the **Userinput.py**, make the following change

Screen of a cell phone

Description automatically generated

1. Type the following in the terminal to check the output of the above code

>>> **python3 Userinput.py**

A picture containing drawing

Description automatically generated

age= int(age) converts the input value to a numerical representation. And after this conversion if you check the type of age, it's displayed as int (integer).

**Python Decision making**

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

**If**

The **if** statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.

Syntax:

**if expression:**

**statement(s)**

If the Boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. If Boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

**else**

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The *else* statement is an optional statement and there could be at most only one **else** statement following **if**.

Syntax:

|  |
| --- |
| **if expression:**  statement(s)  **else:**     statement(s) |

**nested if statements**

There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct.

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Syntax:

|  |
| --- |
| **if** expression1:  statement(s)  **if** expression2:  statement(s)  **elif**expression3:  statement(s)  **elif**expression4:  statement(s)  **else:**  statement(s)  **else:**  statement(s) |

1. Create a **IfControl.py** file and type the following code in the file:

**Note: Python is space sensitive, make sure you have right indentation**

A screenshot of a cell phone

Description automatically generated

1. Open the terminal in the VS Code. Type the following command and give different input to test the program:

>>> **python3 IfControl.py**







**Loops**

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times.

**While loop**

The while loop runs as long as, or while, a certain condition is true.

Syntax:

|  |
| --- |
| **while** expression:     statement(s) |

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are part of a single block of code. Python uses indentation as its method of grouping statements.

1. Create a **WhileControl.py** file and added the code as below:

A screenshot of a cell phone

Description automatically generated

1. In the terminal type the following command:

>>> **python3 WhileControl.py**

A picture containing food, drawing

Description automatically generated

**Infinite Loop- While**

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value.

This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required. Look at the example below

|  |
| --- |
| x = 1  while x <= 5:      print(x) |

The value of x will start at 1 but never change. As a result, the conditional test x <= 5 will always evaluate to True and the while loop will run forever, printing a series of 1s.

If your program gets stuck in an infinite loop, press CTRL-C or just close the terminal window displaying your program's output.

**Using else Statement with while**

When the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

1. Create a file **While-else.py** and type the following code

A screenshot of a cell phone

Description automatically generated

1. In the terminal type the following command:

>>> **python3 While-else.py**

A screenshot of a cell phone

Description automatically generated

The value of count will be incremented each time, and the condition is checked if the count is less than 5. Till when the count is less than 5, the while loop continues and when the condition becomes false, the else statement is executed.

**For loop**

*“for”* loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. The Python *for* statement iterates over the members of a sequence in order, executing the block each time.

Syntax:

|  |
| --- |
| **for** iterating\_var **in** sequence:     statements(s) |

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating\_var. Next, the statements block is executed. Each item in the list is assigned to iterating\_var, and the statement(s) block is executed until the entire sequence is exhausted.

1. Create a **ForControl.py** file and type the code as below:

A picture containing food, drawing

Description automatically generated

Note: we import the random function to generate a random number from 5 to 15 (exclusive) for the end range number.

1. In the terminal type the following command:

>>> **python3 ForControl.py**

A close up of a sign

Description automatically generated

**Iterating by Sequence Index**

An alternative way of iterating through each item is by indexing offset into the sequence itself. Look at the example below.

|  |
| --- |
| fruits = ['banana', 'apple',  'mango']  **for** index **in** range(len(fruits)):  **print** ('Current fruit :', fruits[index])    **print** "Good bye!" |

The above code produces the result

Current fruit : banana

Current fruit : apple

Current fruit : mango

Good bye!

The len() built-in function provides the total number of elements in the tuple as well as the range() built-in function gives us the actual sequence to iterate over. So, the length of the fruit is 3 and the range starts from 0 to 2. When the range is 0 banana is getting printed, range 1- apple is getting printed, range 2- mango is getting printed.

**Using else Statement with for**

When the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.

1. Create a file **ForElse.py** and type the code below.

Screen of a cell phone

Description automatically generated

1. In the terminal type the following command:

>>> **python3 ForElse.py**

A close up of a logo

Description automatically generated

**Loop Control Statements**

Loop control statements change execution from their normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

**break statement:**

It terminates the current loop and resumes execution at the next statement

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

If you are using nested loops, the break statement stops the execution of the innermost loop and starts executing the next line of code after the block.

Syntax:

**break**

1. Create a **Break.py** file and type the code as below:

A close up of text on a screen

Description automatically generated

1. In the terminal type the following command:

>>> **python3 Break.py**

A close up of a logo

Description automatically generated

**continue statement:**

It returns the control to the beginning of the loop. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

Syntax:

**continue**

1. Create a **Continue.py** file and type the code as below:

A close up of text on a screen

Description automatically generated

1. In the terminal type the following command:

>>> **python3 Continue.py**

A picture containing computer

Description automatically generated

**Data conversion**

1. Create a **DataConversion.py** file and type the following code in the file:

A black sign with white text

Description automatically generated

Note: we want to output the total price based on the number a user gives.

1. In the VS Code terminal type the following command to run the program:

>>> **python3 DataConversion.py**



Surprisingly, we got a very expensive bill. This is because Python always gets the user input as a string and a number \* a string will cause the system to print the string many times based on the number.

1. We can fix the issue through explicitly converting the data type to what we want. Update the code as below:

A picture containing monitor, black, television, screen

Description automatically generated

1. In the VS Code terminal type the following command:

>>> **python3 DataConversion.py**

A close up of a sign

Description automatically generated

Note: we use int() to convert the input value to an integer and use str() to convert the result of calculation back to a string. We also can use float() to convert a value to a float number.