# AI 620 Emerging Topics in Artificial Intelligence
# HOS01A-2 – Python Basics

03/01/2020 Developed by Amrutha Vaidyanathan

09/10/2024 Reviewed by Naveena Moddu

09/14/2024 Reviewed by Anh Nguyen

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

**Before You Start**

- The HOS assignment is 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, consult the TA.

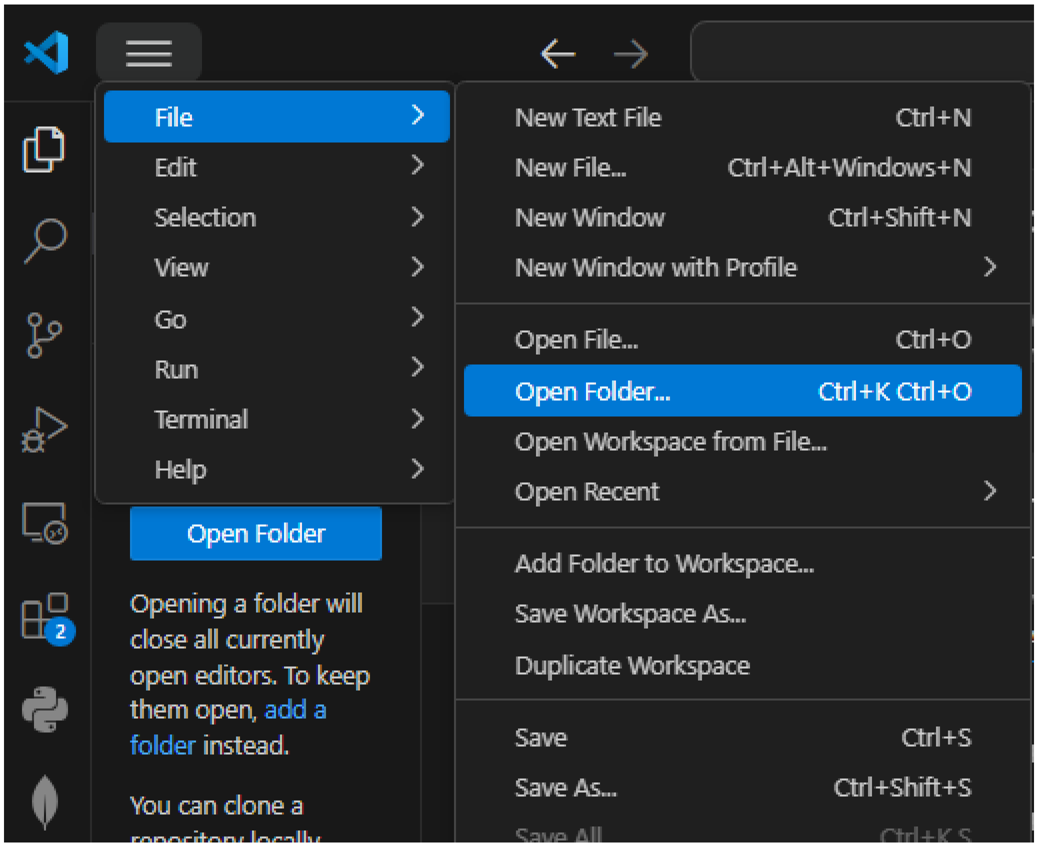
**Learning Outcomes**

Students will be able to:

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

## Set up

1. Open VS code and then open the course's folder.



2. In the VS code terminal type the commands for navigating to the **Module0** folder

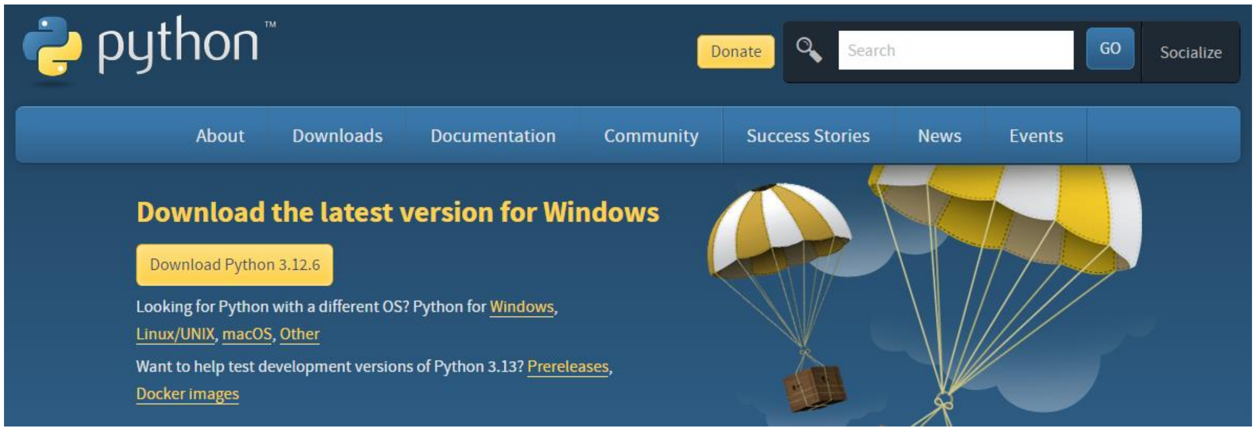
```
cd Module0
```

## Install Python on 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/

2. Click the downloading link based on the operational system you are using:


python™
Donate
Search
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
Download the latest version for Windows
Download Python 3.12.6
Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other
Want to help test development versions of Python 3.13? Prereleases, Docker images

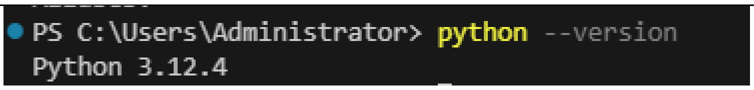
Note: When this document is created, the latest Python version was 3.12.6, you might see a higher version number. Please download the latest version.

3. Perform the installation process:

   Click on **Download Python 3.12.6** (you might see a different version). If you are using Windows, the **.exe** file will get downloaded. If you are using a Mac, **.pkg** file will get downloaded. After downloading the files, follow the instructions on your screen to install python.

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

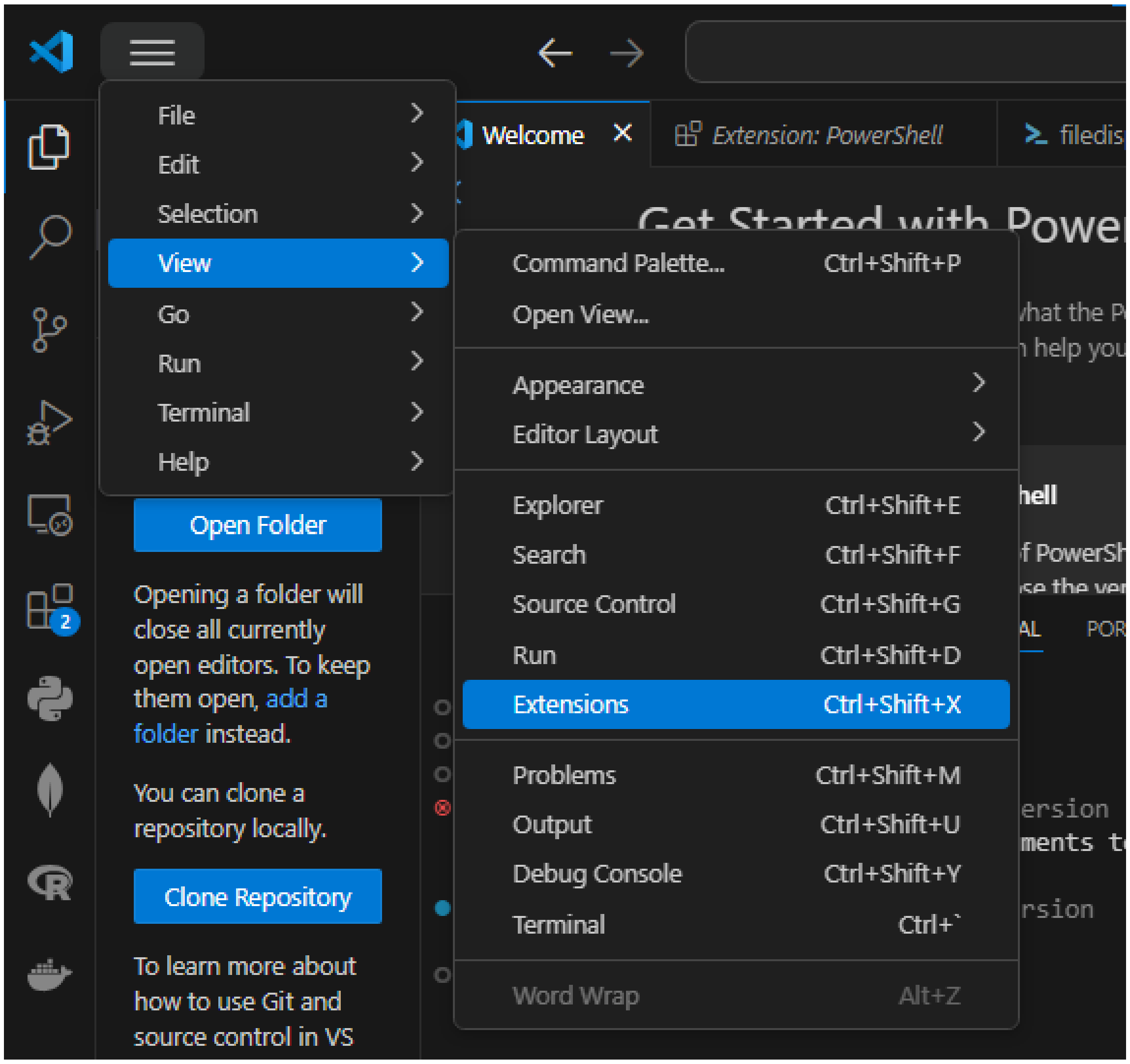
```
python --version
```



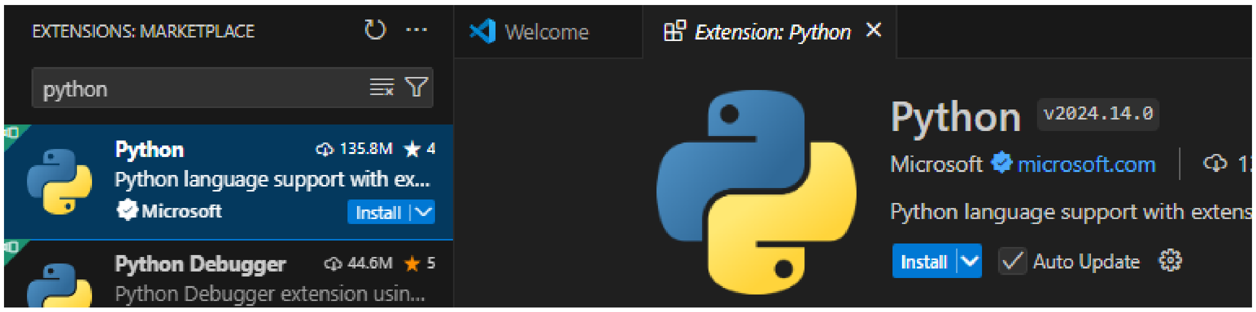
```
PS C:\Users\Administrator> python --version
Python 3.12.4
```

Note: You might see a different python version (It may be slightly different from the version shown here. But it should be 3.x.x).

5. The Python extension is also recommended to install in the VS Code.
   Click the **View** => **Extension** in the VS Code menu.



Search python and click the install button on the right



# 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**

```
message = “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.

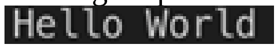
   To display the contents inside the message use 'print' function.

```
print()
```

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

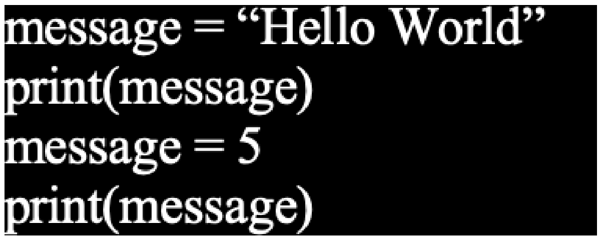
```
python3 filename.py
```

   The above code will display the following output.



```
Hello World
```

2. Look at the example below to understand how variables work. You don't have to type the code.



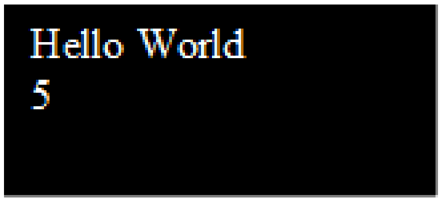
```
message = “Hello World”
print(message)
message = 5
print(message)
```

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

The variable ‘message’ has been used twice and the most 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

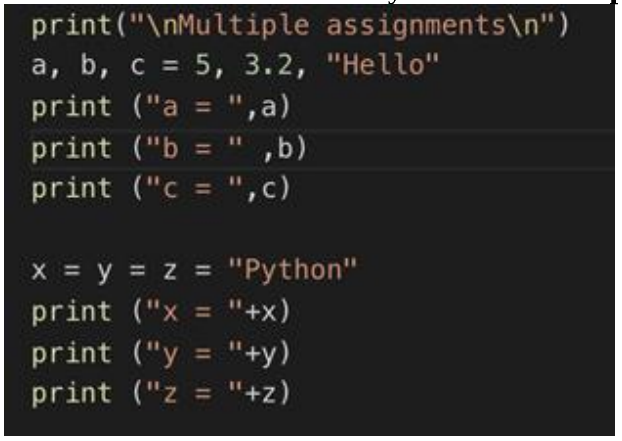


```
Hello World
5
```

## Multiple assignments

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

1. Under Module1 create a file by name **variable.py** and type the following code.



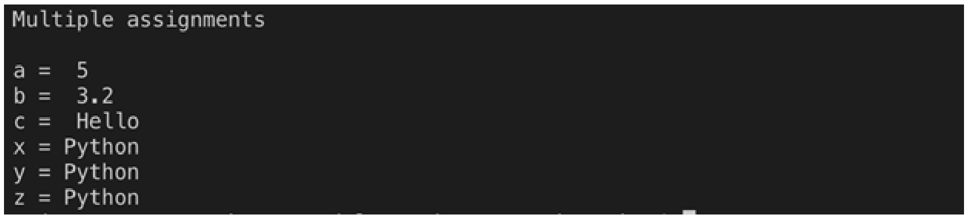
```
print("\nMultiple assignments\n")
a, b, c = 5, 3.2, "Hello"
print ("a = ",a)
print ("b = " ,b)
print ("c = ",c)

x = y = z = "Python"
print ("x = "+x)
print ("y = "+y)
print ("z = "+z)
```

2. Before checking the output for the previous program check if you are in the correct directory.

   You must be under ../**Module1**
   To check the path, you can type “`pwd`” in the terminal.

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

```
python3 variables.py
```



```
Multiple assignments

a =  5
b =  3.2
c =  Hello
x = Python
y = Python
z = Python
```

## 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 capital. 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

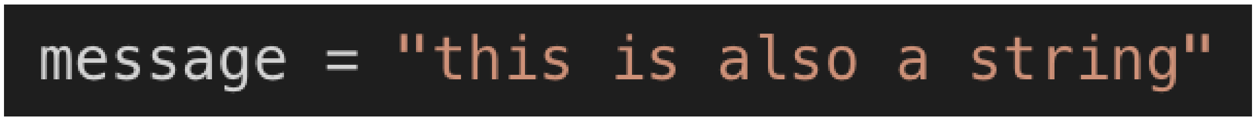
A dictionary object is an unordered collection of data in a key: value pair form. A collection of such pairs is enclosed in curly brackets. For example:

```
{1:"Steve", 2:"Bill", 3:"Ram", 4: "Farha"}
```

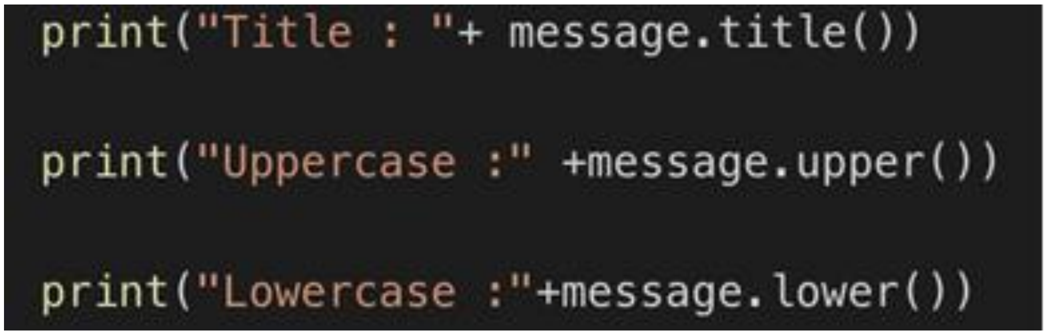
## String functions

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

1) Under Module1, create another file **strings.py** and type the following code.



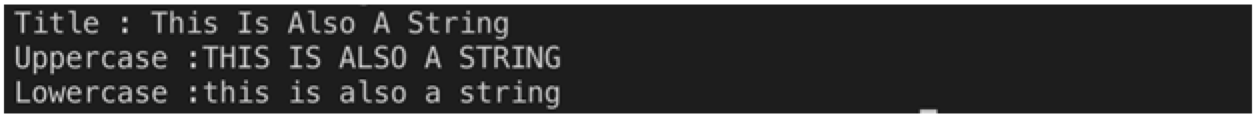
```
message = "this is also a string"
```



```
print("Title : "+ message.title())

print("Uppercase :" +message.upper())

print("Lowercase :"+message.lower())
```

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

```
python3 strings.py
```



```
Title : This Is Also A String
Uppercase :THIS IS ALSO A STRING
Lowercase :this is also a string
```

**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.

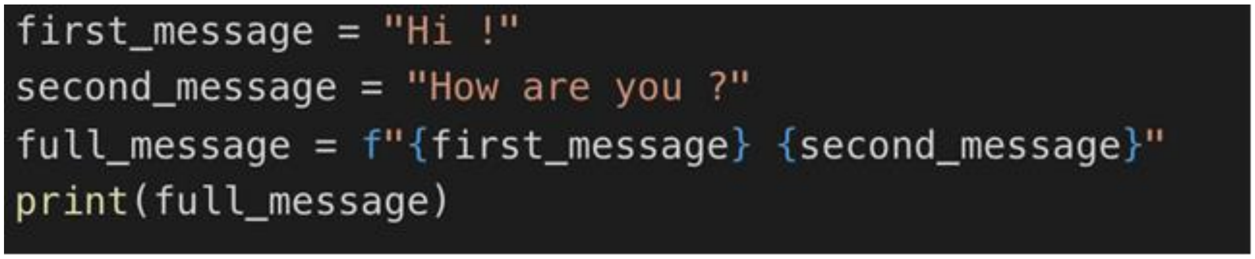
The `upper()` method returns a string where all characters are in upper case. Symbols and Numbers are ignored.

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.

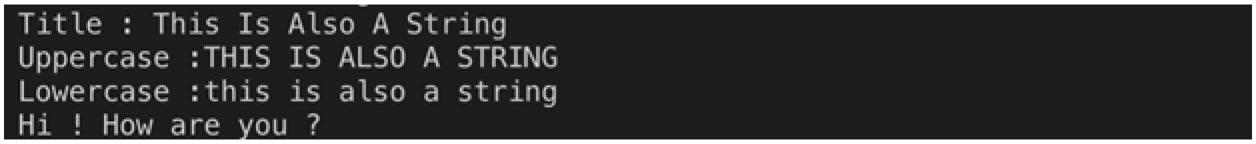
3) In the same **strings.py**, add the code below.



```
first_message = "Hi !"
second_message = "How are you ?"
full_message = f"{first_message} {second_message}"
print(full_message)
```

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

```
python3 Strings.py
```



```
Title : This Is Also A String
Uppercase :THIS IS ALSO A STRING
Lowercase :this is also a string
Hi ! How are you ?
```

**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

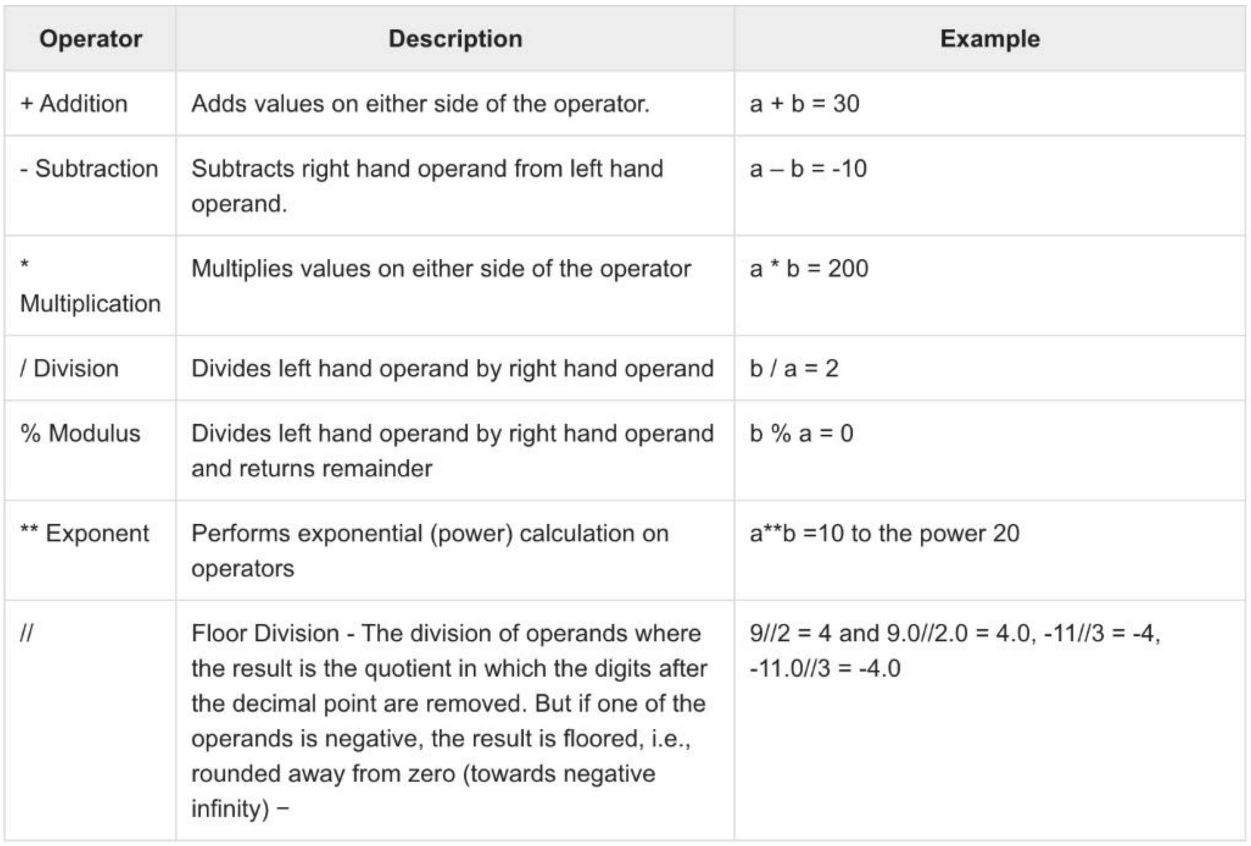
Python language supports the following types of operators.

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

The commonly used operators are discussed below.

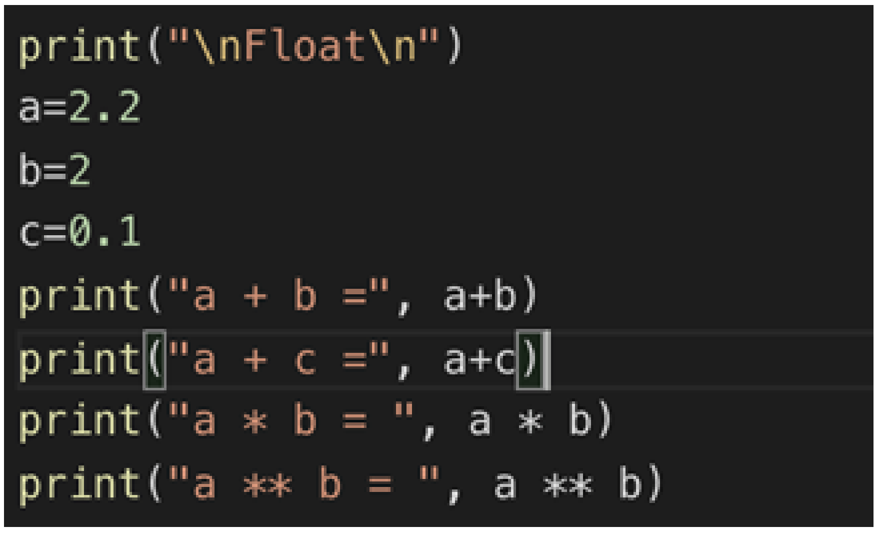
**Arithmetic operator:**

Consider value of a = 10, b = 20



| Operator | Description | Example |
|---|---|---|
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |
| * Multiplication | Multiplies values on either side of the operator | a * b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| ** Exponent | Performs exponential (power) calculation on operators | a**b =10 to the power 20 |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) − | 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0 |

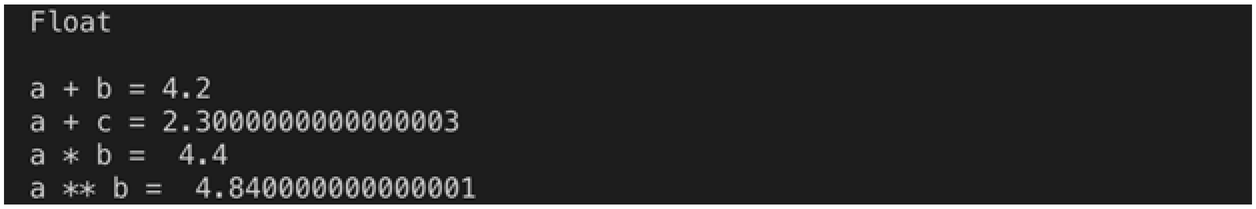
1) Under module1, create a file **numbers.py** and type the following code.



```
print("\nFloat\n")
a=2.2
b=2
c=0.1
print("a + b =", a+b)
print("a + c =", a+c)
print("a * b = ", a * b)
print("a ** b = ", a ** b)
```

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

```
python3 numbers.py
```

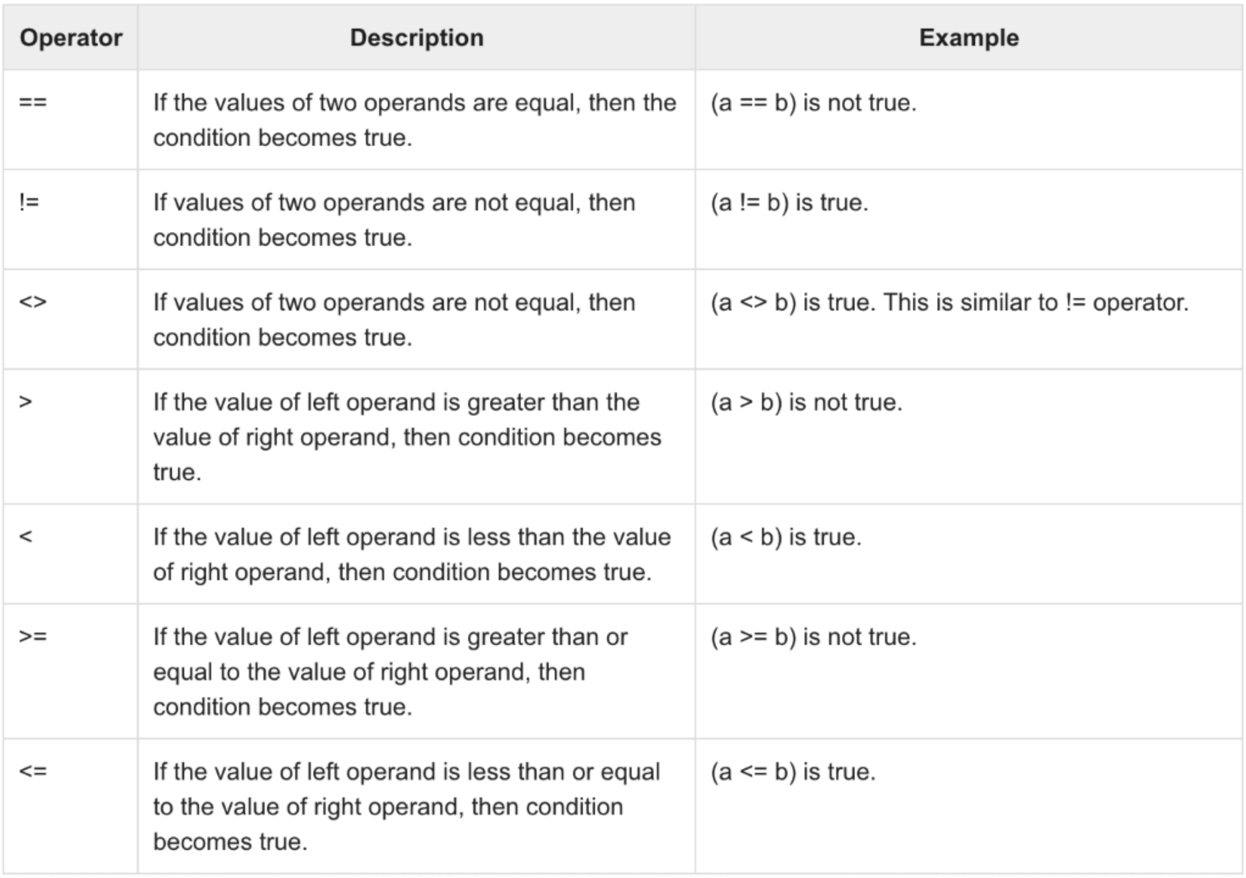


```
Float

a + b = 4.2
a + c = 2.3000000000000003
a * b =  4.4
a ** b =  4.840000000000001
```

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

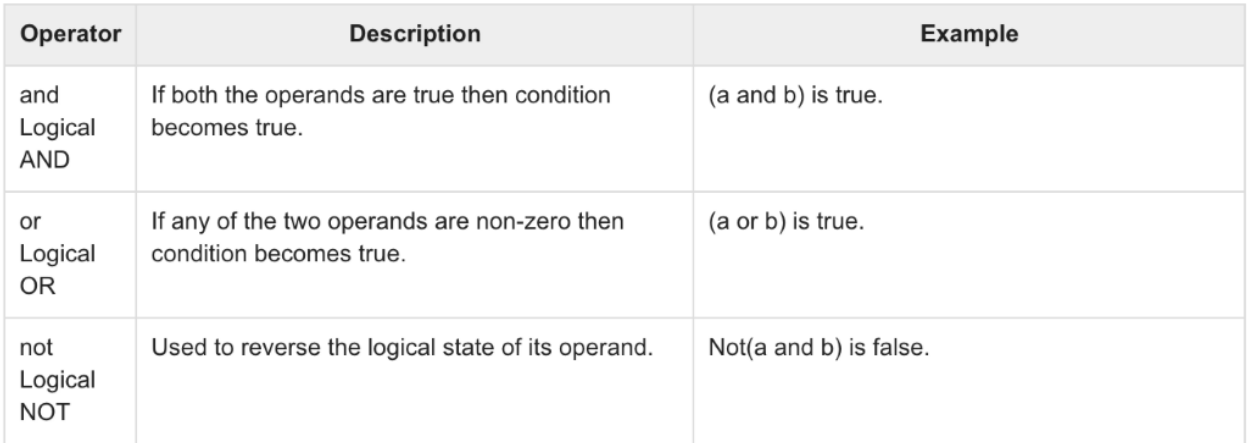
Consider value of a = 10, b = 20



| Operator | Description | Example |
|---|---|---|
| == | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

## Logical Operator

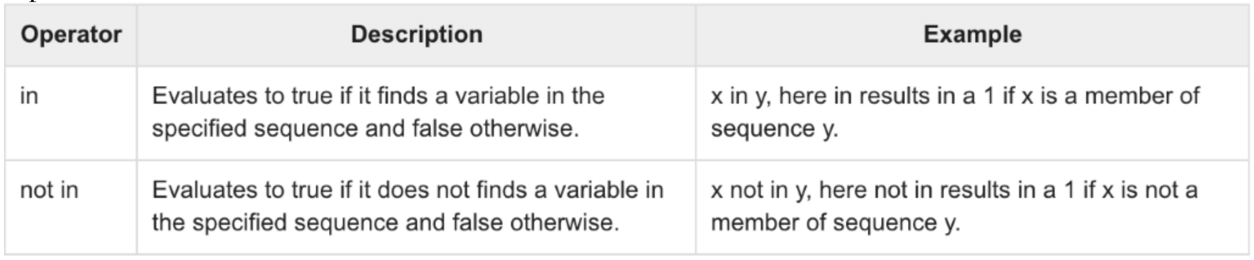
Consider value of a = 10, b = 20



| Operator | Description | Example |
|---|---|---|
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

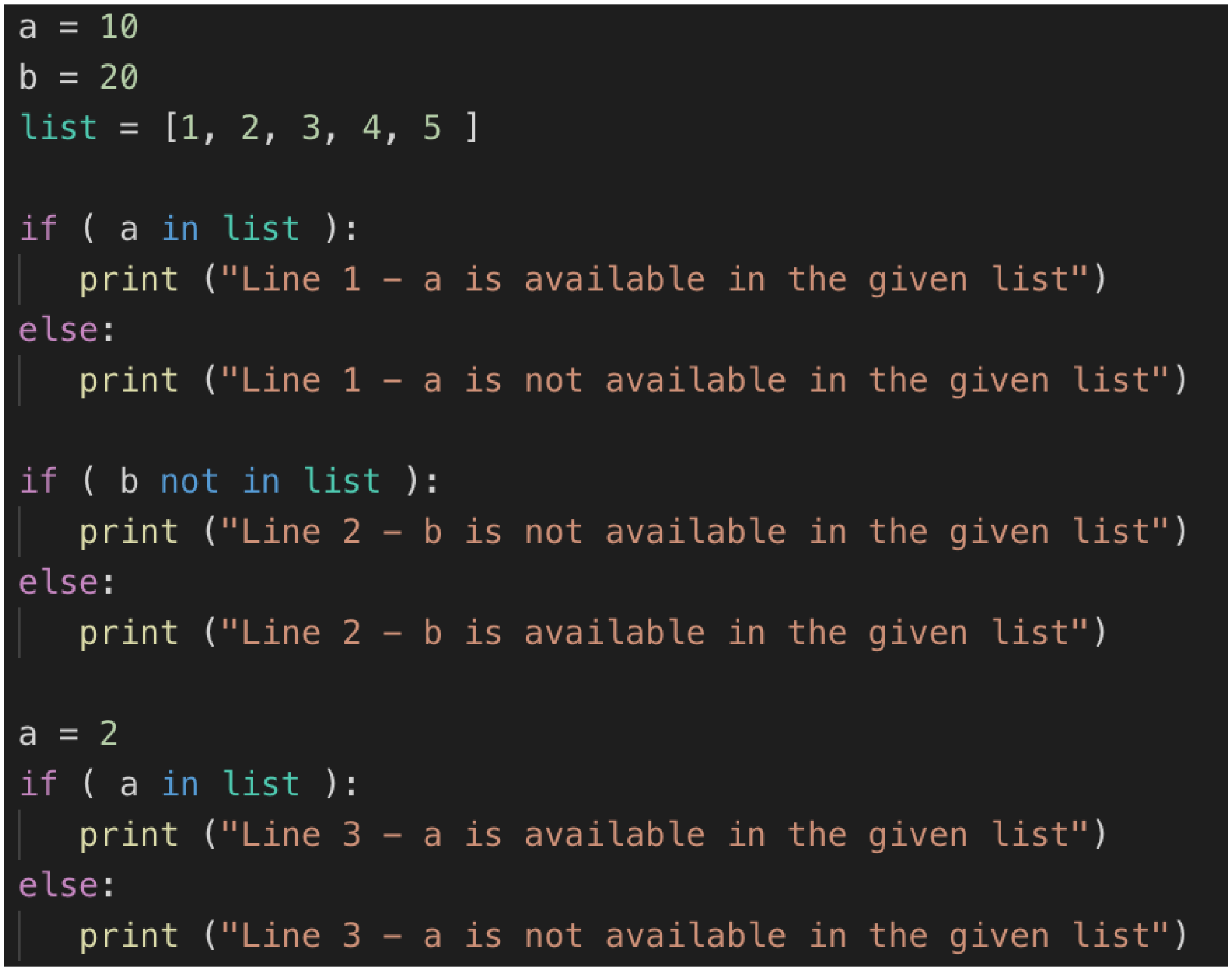
## Membership Operator

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



| Operator | Description | Example |
|---|---|---|
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

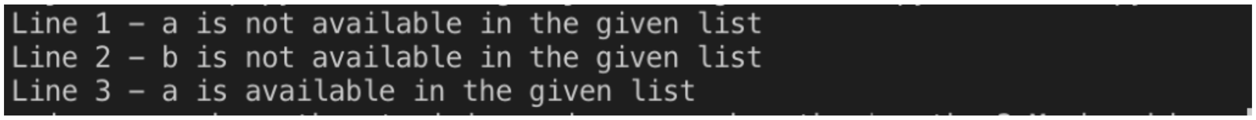
1) Under Module1, create a file **membership.py** and type the following code.



```python
a = 10
b = 20
list = [1, 2, 3, 4, 5 ]

if ( a in list ):
    print ("Line 1 - a is available in the given list")
else:
    print ("Line 1 - a is not available in the given list")

if ( b not in list ):
    print ("Line 2 - b is not available in the given list")
else:
    print ("Line 2 - b is available in the given list")

a = 2
if ( a in list ):
    print ("Line 3 - a is available in the given list")
else:
    print ("Line 3 - a is not available in the given list")
```

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

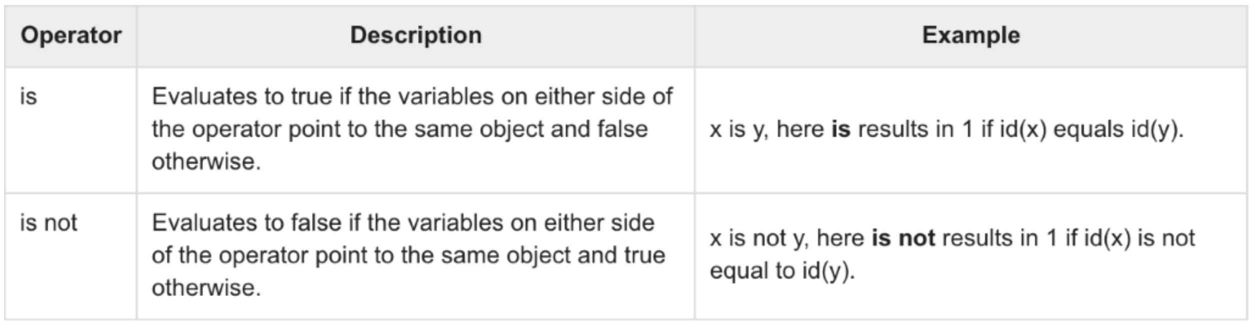
```
python3 Membership.py
```



```
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

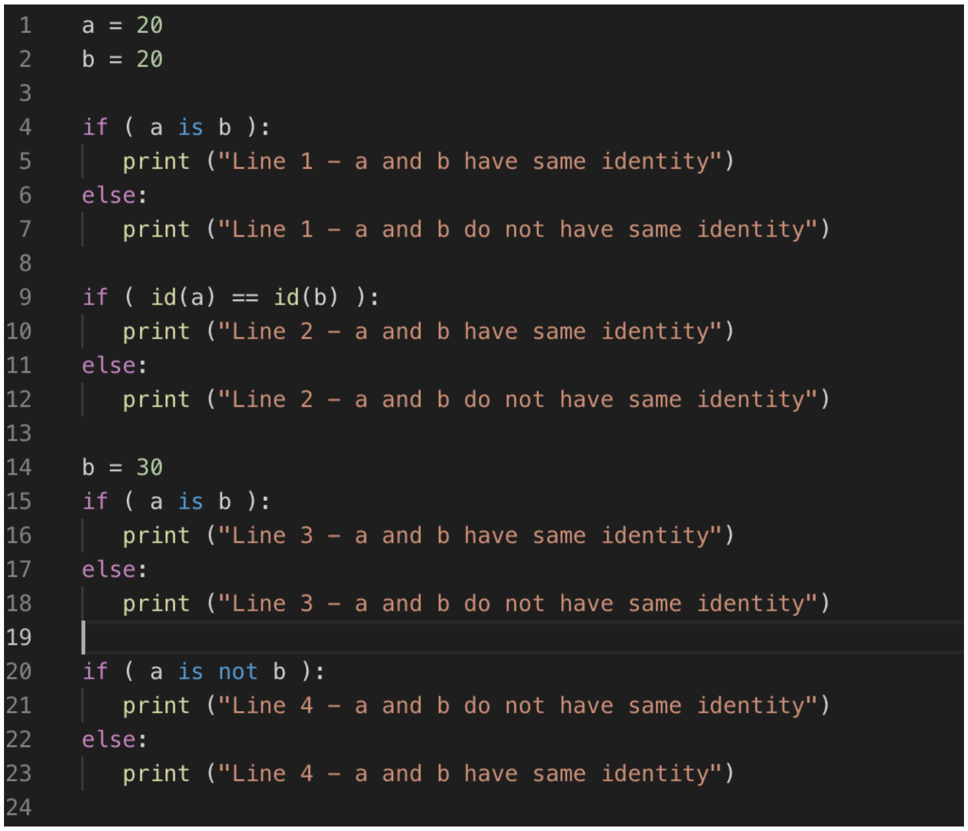
## Identity Operator

Identity operators compare the memory locations of two objects.



| Operator | Description | Example |
|---|---|---|
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here **is** results in 1 if id(x) equals id(y). |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here **is not** results in 1 if id(x) is not equal to id(y). |

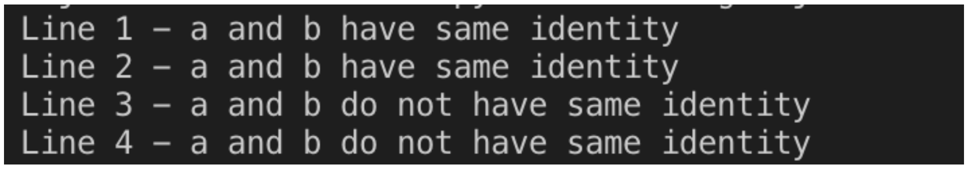
1) Under Module1, create a file **identity.py** and type the following code.



```python
a = 20
b = 20

if ( a is b ):
    print ("Line 1 - a and b have same identity")
else:
    print ("Line 1 - a and b do not have same identity")

if ( id(a) == id(b) ):
    print ("Line 2 - a and b have same identity")
else:
    print ("Line 2 - a and b do not have same identity")

b = 30
if ( a is b ):
    print ("Line 3 - a and b have same identity")
else:
    print ("Line 3 - a and b do not have same identity")

if ( a is not b ):
    print ("Line 4 - a and b do not have same identity")
else:
    print ("Line 4 - a and b have same identity")

```

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

```
python3 identity.py
```



```
Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

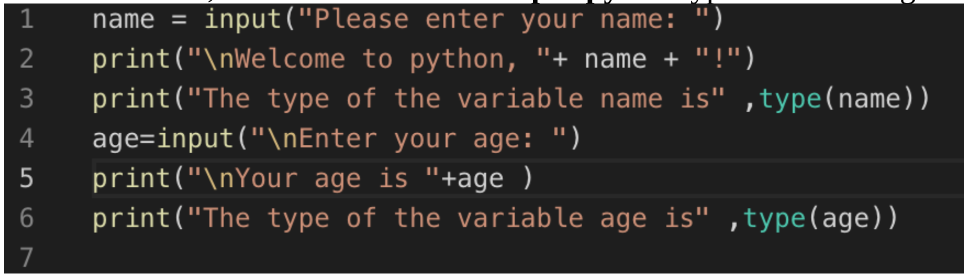
## User Input

To receive information through the keyboard, Python uses the `input()` function. These functions have 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()` function 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 be submitted to the application.

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

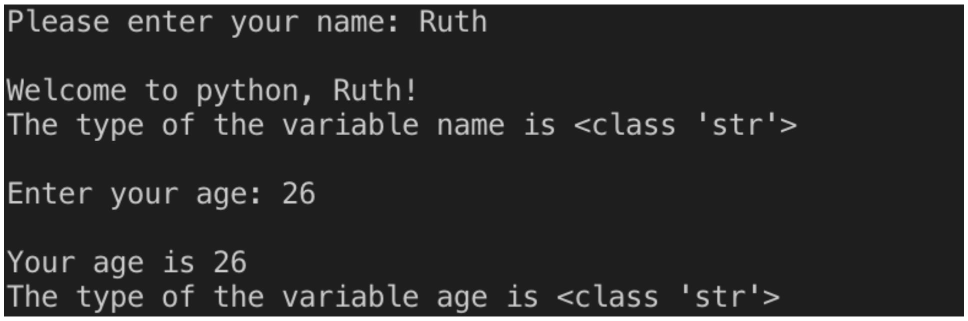
1. Under Module1, create a new file **userInput.py** and type the following code



```python
name = input("Please enter your name: ")
print("\nWelcome to python, "+ name + "!")
print("The type of the variable name is" ,type(name))
age=input("\nEnter your age: ")
print("\nYour age is "+age )
print("The type of the variable age is" ,type(age))
```

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

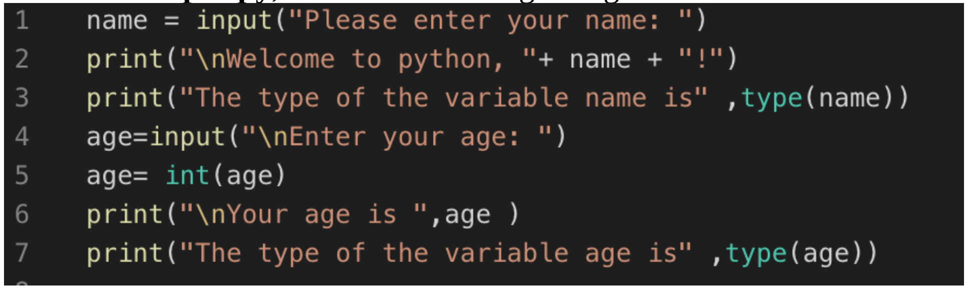
```
python3 userInput.py
```



```
Please enter your name: Ruth

Welcome to python, Ruth!
The type of the variable name is <class 'str'>

Enter your age: 26

Your age is 26
The type of the variable age is <class 'str'>
```

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.

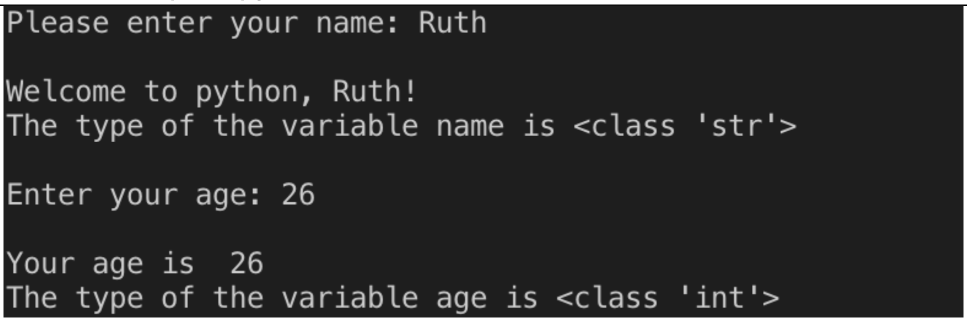
3. In the **userInput.py**, make the following change



```python
name = input("Please enter your name: ")
print("\nWelcome to python, "+ name + "!")
print("The type of the variable name is" ,type(name))
age=input("\nEnter your age: ")
age= int(age)
print("\nYour age is ",age )
print("The type of the variable age is" ,type(age))
```

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

```
python3 userInput.py
```



```
Please enter your name: Ruth

Welcome to python, Ruth!
The type of the variable name is <class 'str'>

Enter your age: 26

Your age is  26
The type of the variable age is <class 'int'>
```

`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.

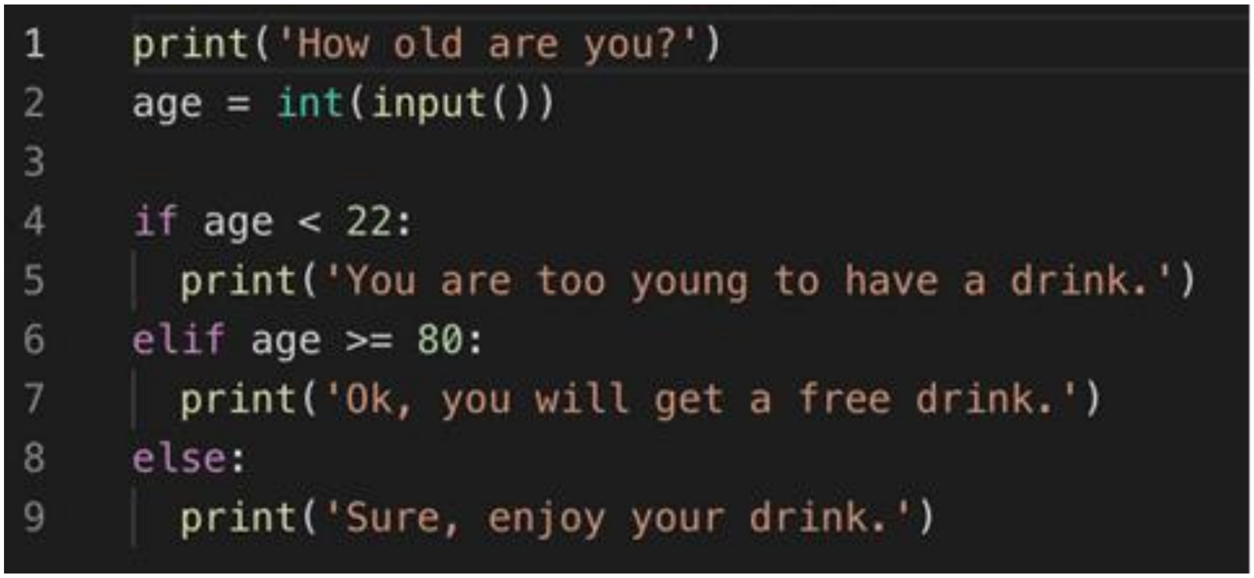
In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

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 under the Module1 folder and type the following code in the file:

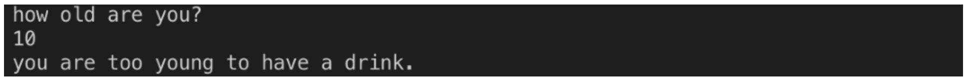
**Note: Python is space sensitive, make sure you have right indentation**



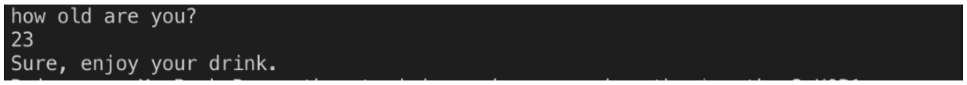
```
print('How old are you?')
age = int(input())

if age < 22:
  print('You are too young to have a drink.')
elif age >= 80:
  print('Ok, you will get a free drink.')
else:
  print('Sure, enjoy your drink.')
```

2. Open the terminal in the VS Code and navigate to the Module1 folder. Type the following command and give different input to test the program:

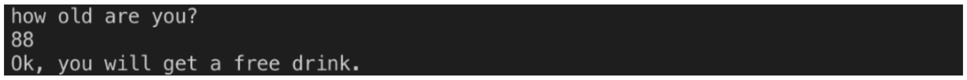
```
python3 ffControl.py
```



```
how old are you?
10
you are too young to have a drink.
```



```
how old are you?
23
Sure, enjoy your drink.
```



```
how old are you?
88
Ok, you will get a free drink.
```

## 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

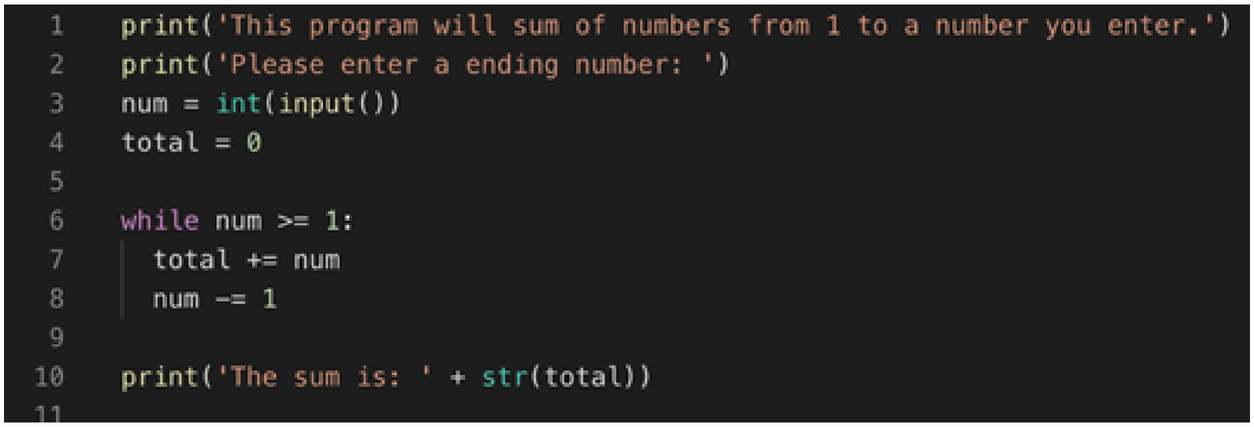
The while loop runs as long as, or while, a certain condition is true.

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.

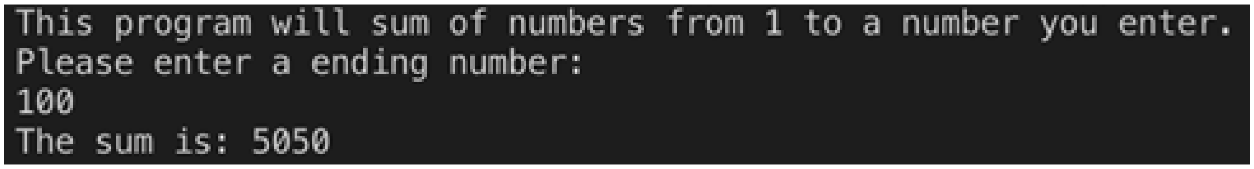
1. Create a **whileControl.py** file under the Module1 folder and add the code as below:



```python
print('This program will sum of numbers from 1 to a number you enter.')
print('Please enter a ending number: ')
num = int(input())
total = 0

while num >= 1:
    total += num
    num -= 1

print('The sum is: ' + str(total))
```

2. In the terminal type the following command:

```
python3 whileControl.py
```



```
This program will sum of numbers from 1 to a number you enter.
Please enter a ending number:
100
The sum is: 5050
```

## Infinite Loop - While

A loop becomes an 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

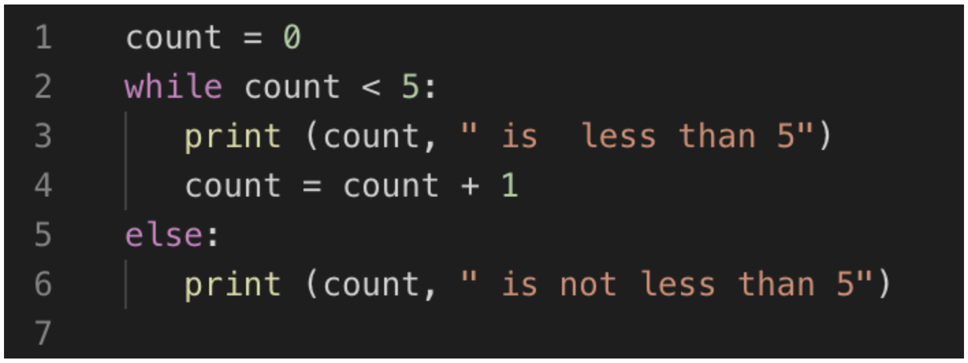
```python
x = 1
while x <= 5:
    print(x)
```

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

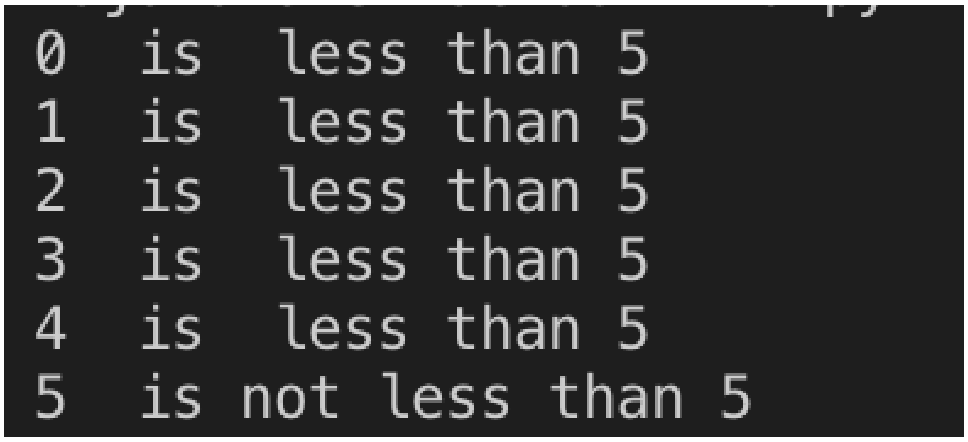
When the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

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


```python
count = 0
while count < 5:
    print (count, " is  less than 5")
    count = count + 1
else:
    print (count, " is not less than 5")
```

2. In the terminal type the following command:

```
python3 whileElse.py
```


```
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

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 continuous and when the condition becomes false, in the else part the statement is executed.

## For loop

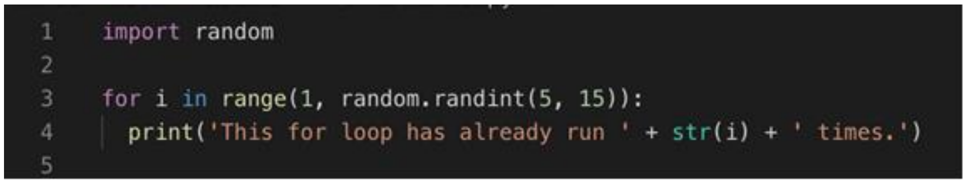
*"for"* loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. The Python *for* statement iterates over the members of a sequence in order, executing the block each time.

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:

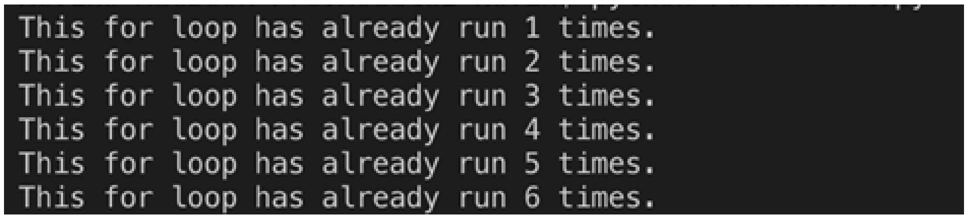


```python
import random

for i in range(1, random.randint(5, 15)):
    print('This for loop has already run ' + str(i) + ' times.')

```

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

2. In the terminal type the following command:

```
python3 forControl.py
```



```
This for loop has already run 1 times.
This for loop has already run 2 times.
This for loop has already run 3 times.
This for loop has already run 4 times.
This for loop has already run 5 times.
This for loop has already run 6 times.
```

## Iterating by Sequence Index

An alternative way of iterating through each item is by index 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 "Goodbye!"
```

The above code produces the result

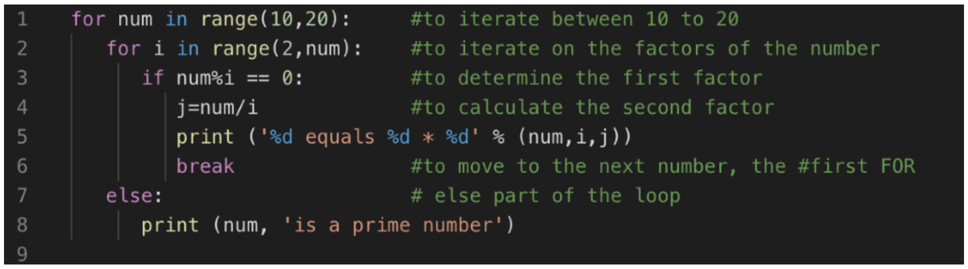
```
Current fruit: banana
Current fruit: apple
Current fruit: mango
Goodbye!
```

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.

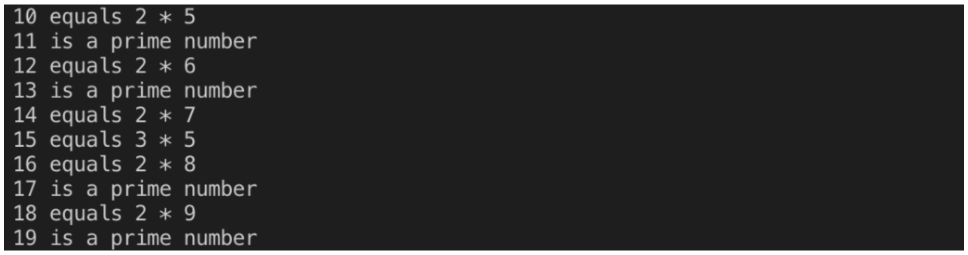
1. Create a file **forElse.py under** Module1 and type the code as below.



```
for num in range(10,20):      #to iterate between 10 to 20
    for i in range(2,num):    #to iterate on the factors of the number
        if num%i == 0:        #to determine the first factor
            j=num/i           #to calculate the second factor
            print ('%d equals %d * %d' % (num,i,j))
            break             #to move to the next number, the #first FOR
    else:                     # else part of the loop
        print (num, 'is a prime number')
```

2. In the terminal type the following command:

```
python3 forElse.py
```



```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

## 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:**

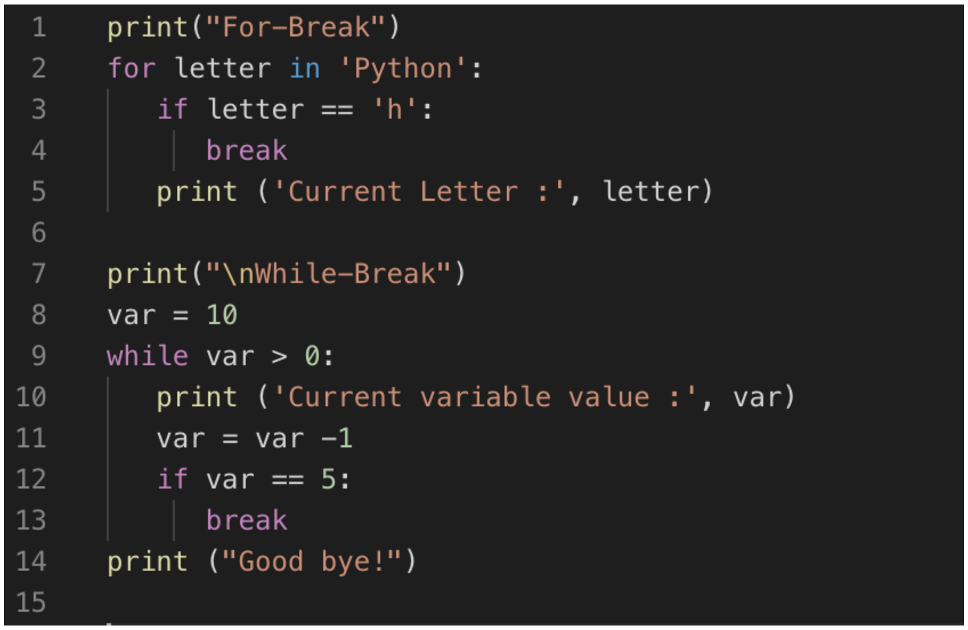
It terminates the current loop and resumes execution at the next statement
The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

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

Syntax:

```
break
```

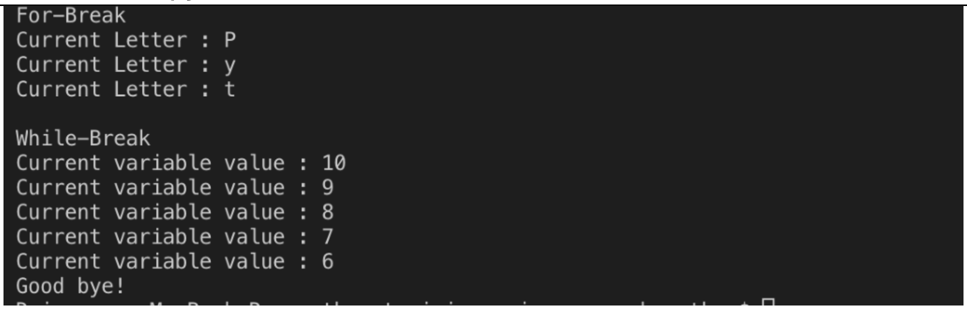
1. Create a **break.py** file under the Module1 folder and type the code as below:



```python
print("For-Break")
for letter in 'Python':
    if letter == 'h':
        break
    print ('Current Letter :', letter)

print("\nWhile-Break")
var = 10
while var > 0:
    print ('Current variable value :', var)
    var = var -1
    if var == 5:
        break
print ("Good bye!")

```

2. In the terminal type the following command:

```
python3 Break.py
```



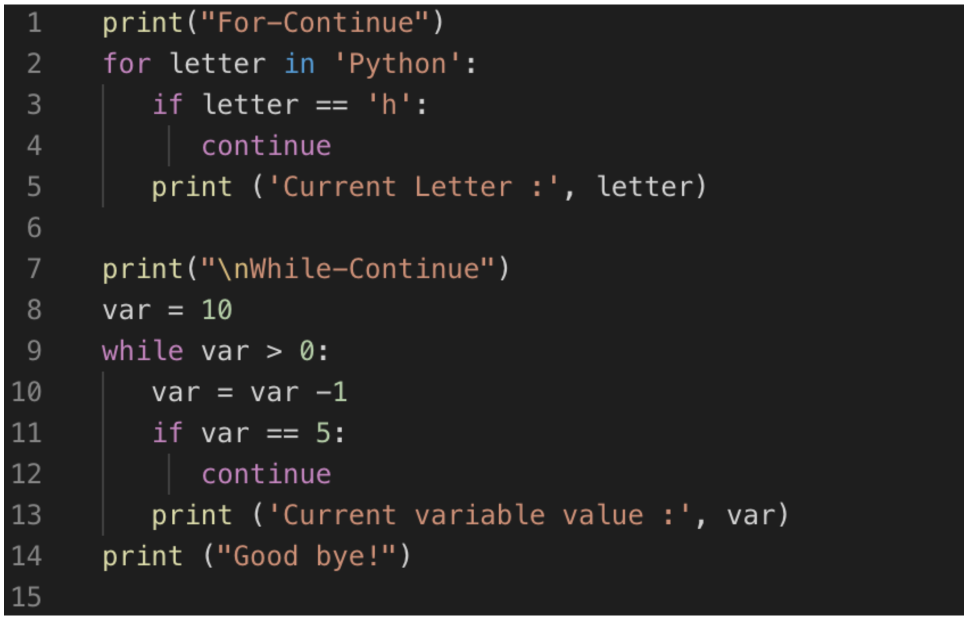
```
For-Break
Current Letter : P
Current Letter : y
Current Letter : t

While-Break
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

**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
```

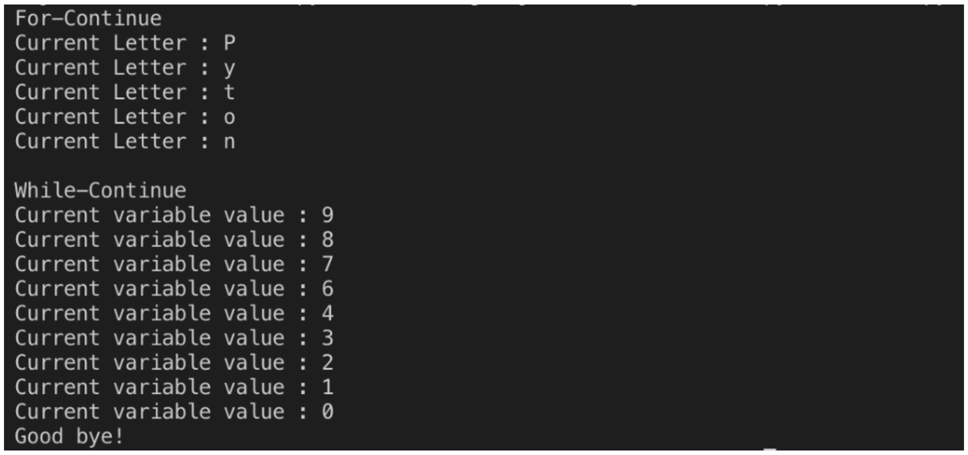
1. Create a **continue.py** file under Module1 folder and type the code as below:



```
print("For-Continue")
for letter in 'Python':
    if letter == 'h':
        continue
    print ('Current Letter :', letter)

print("\nWhile-Continue")
var = 10
while var > 0:
    var = var -1
    if var == 5:
        continue
    print ('Current variable value :', var)
print ("Good bye!")
```

2. In the terminal type the following command:

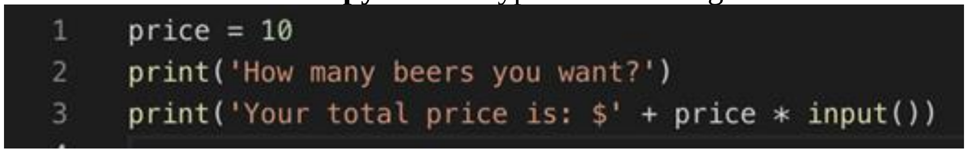
```
python3 continue.py
```



```
For-Continue
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n

While-Continue
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

## Data conversion

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

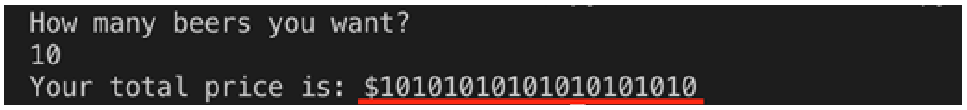


```
price = 10
print('How many beers you want?')
print('Your total price is: $' + price * input())
```

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

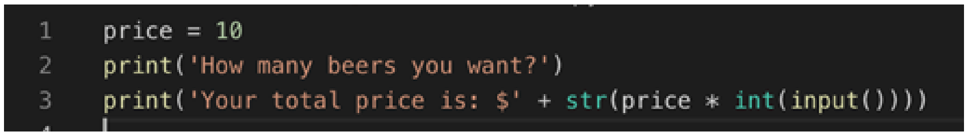
2. In the VS Code terminal type the following command to run the program:

```
python3 dataConversion.py
```


```
How many beers you want?
10
Your total price is: $10101010101010101010
```

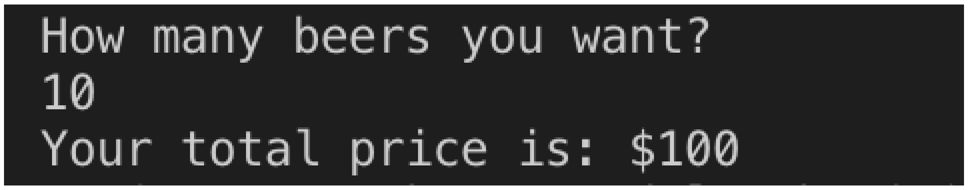
Surprisingly, we got a very expensive bill. This is because Python always gets the user input as a string and a number * a string will cause the system to print the string many times based on the number.

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


```python
price = 10
print('How many beers you want?')
print('Your total price is: $' + str(price * int(input())))
```

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

```
python3 dataConversion.py
```


```
How many beers you want?
10
Your total price is: $100
```

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.