Basics of Python Programming-Fundamentals-I

Features and History of Python

- It is a high-level, interpreted, interactive, object-oriented and a reliable language. i.e. very simple and uses English-like words.
- It has a vast library of modules to support integration of complex solutions from pre-built components.
- Python is an open-source project, supported by many individual.
- It is platform independent, scripted language.
- It allows users to integrate applications seamlessly to create high-powered, highly-focused applications.

Thus python is a complete programming language with the following features:

Simple,	Easy to learn
Versatile,	Free & open source
High level language,	Interactive
Portable,	Object-oriented
Interpreted,	Dynamic
Extensible,	Embeddable
Extensive libraries,	Easy Maintenance
Secure,	Robust
Multi-threaded,	Garbage collection

Applications of Python

Python is a high-level general purpose programming language that is used to develop a wide range of applications including image processing, text processing, web, enterprise level applications using scientific an numeric data from network.

Applications are:

Embedded scripting language, 3D software, Web development, GUI based desktop applications, Image processing & GUI, Games Scientific & computational applications, Teaching Operating systems, Language development Prototyping, Network programming

Python Features

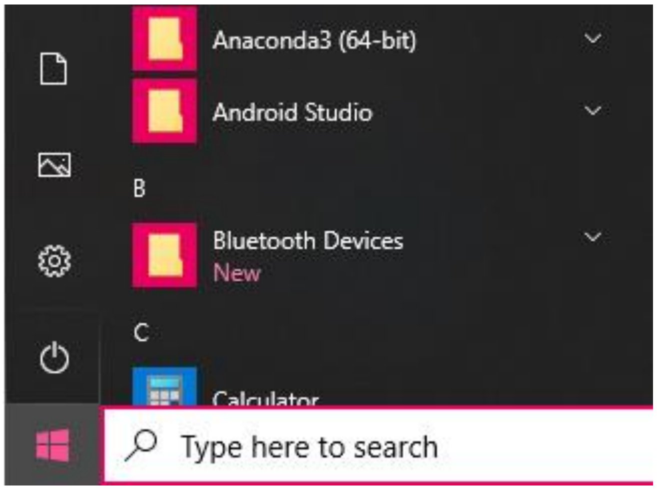
- Beginner's Language
- Simple and Easy to Learn
- Interpreted Language
- Cross-platform language
- Free and Open Source
- Object-Oriented language
- Extensive Libraries
- Integrated
- Databases Connectivity

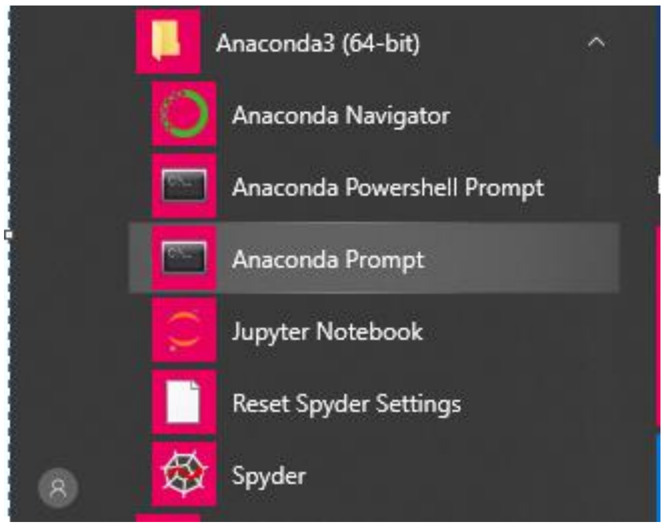
==

You all are well aware of above topics about introduction to Python, now we can start our journey into coding part in python with the help of Jupyter Notebook.

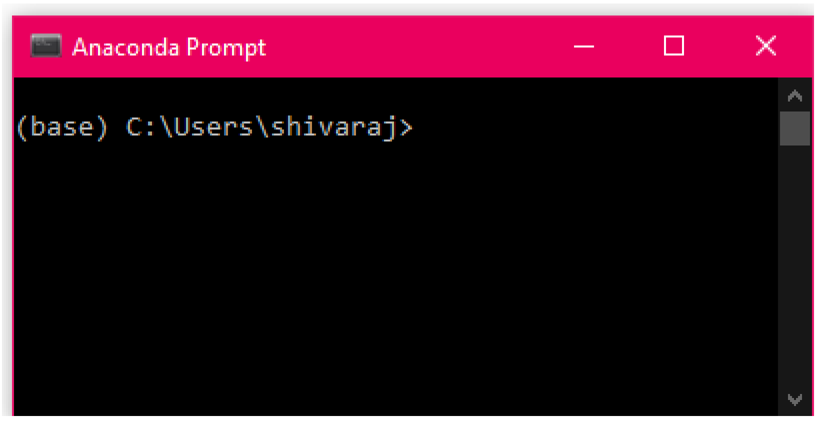
Jupyter Notebook: is browser based interpreter that allows us to interactively work with python.

- To work with Jupyter Notebook you need to install Anaconda Software, but you have already installed.
- Now go to Start, select Anaconda Folder, and select Drop down menu and Click on Anaconda Prompt.



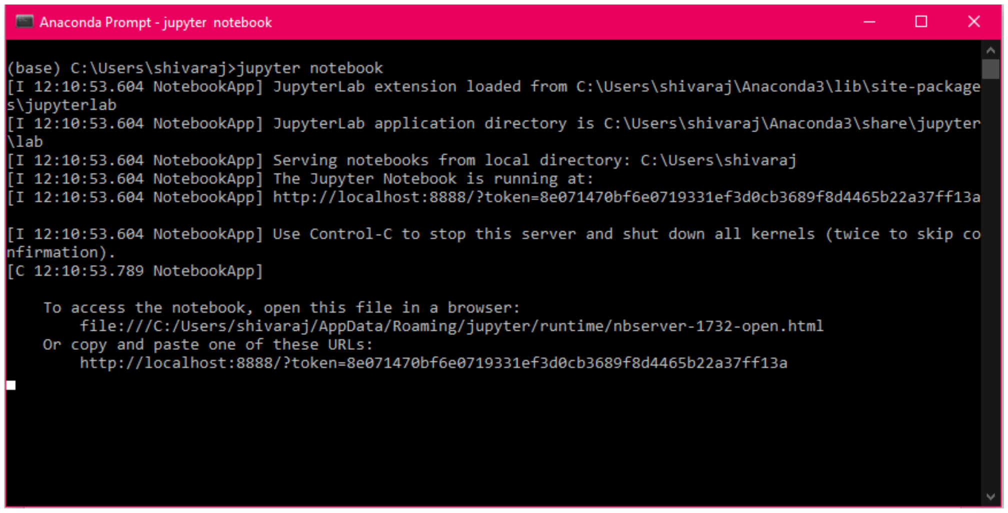


- Once you done with this you will get the below window as Anaconda Prompt



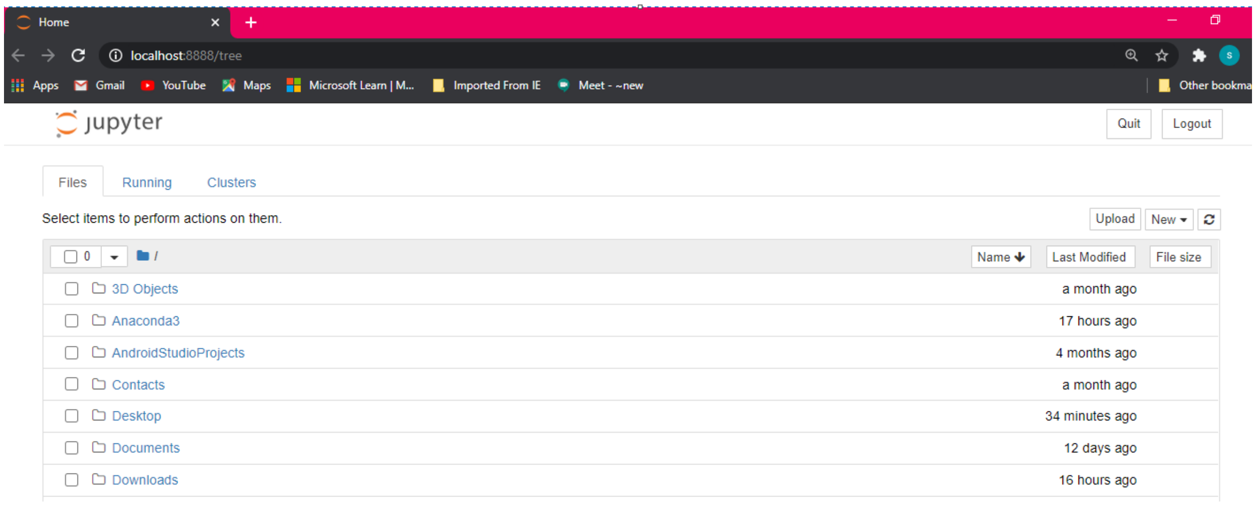
```
(base) C:\Users\shivaraj>
```

- In the command prompt type Jupyter notebook and wait for a while as it loads from the main Anaconda package and it looks like below image.



```
(base) C:\Users\shivaraj>jupyter notebook
[I 12:10:53.604 NotebookApp] JupyterLab extension loaded from C:\Users\shivaraj\Anaconda3\lib\site-package
s\jupyterlab
[I 12:10:53.604 NotebookApp] JupyterLab application directory is C:\Users\shivaraj\Anaconda3\share\jupyter
\lab
[I 12:10:53.604 NotebookApp] Serving notebooks from local directory: C:\Users\shivaraj
[I 12:10:53.604 NotebookApp] The Jupyter Notebook is running at:
[I 12:10:53.604 NotebookApp] http://localhost:8888/?token=8e071470bf6e0719331ef3d0cb3689f8d4465b22a37ff13a

[I 12:10:53.604 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip co
nfirmation).
[C 12:10:53.789 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/shivaraj/AppData/Roaming/jupyter/runtime/nbserver-1732-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=8e071470bf6e0719331ef3d0cb3689f8d4465b22a37ff13a
```

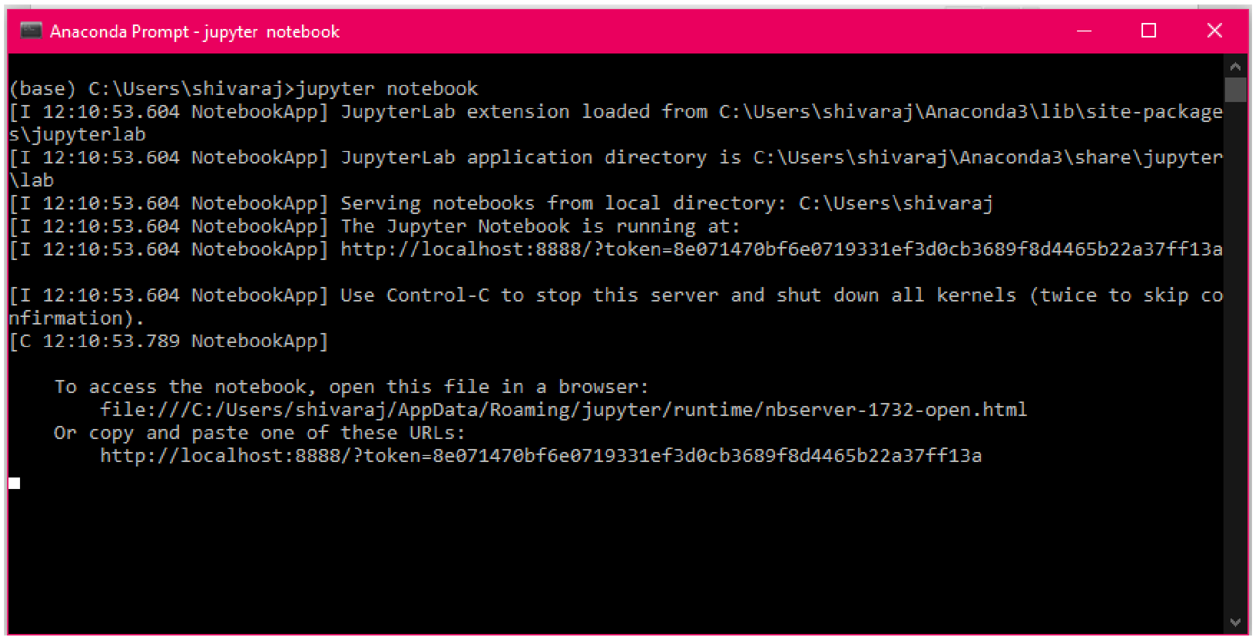
- Once this is done your web browser is automatically opens with below image and keep in mind it is a browser based interaction



Select items to perform actions on them.

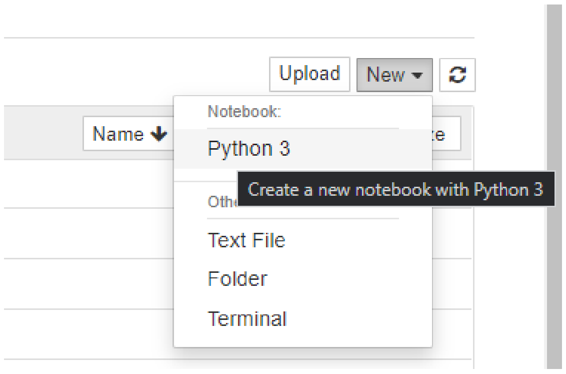
	Name	Last Modified
☐	3D Objects	a month ago
☐	Anaconda3	17 hours ago
☐	AndroidStudioProjects	4 months ago
☐	Contacts	a month ago
☐	Desktop	34 minutes ago
☐	Documents	12 days ago
☐	Downloads	16 hours ago

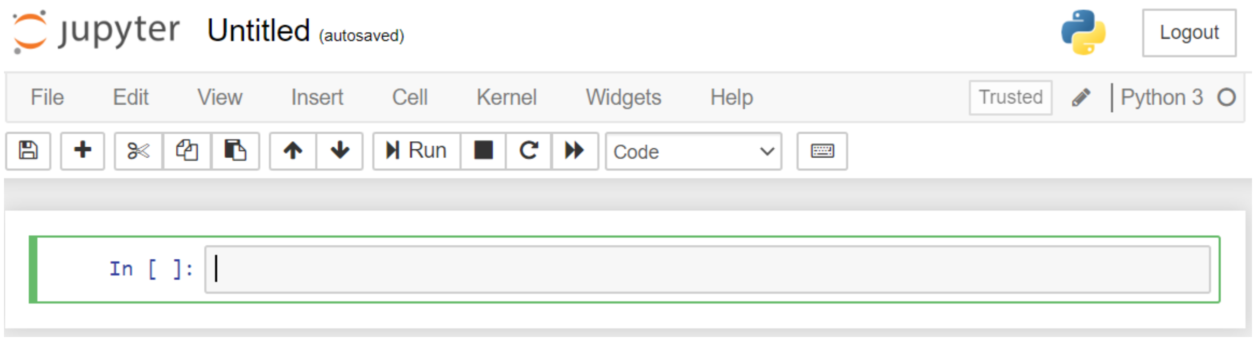
- Now your Jupyter notebook is up and running, and this can be confirmed by switching into Anaconda Prompt and you can see the below information in that window



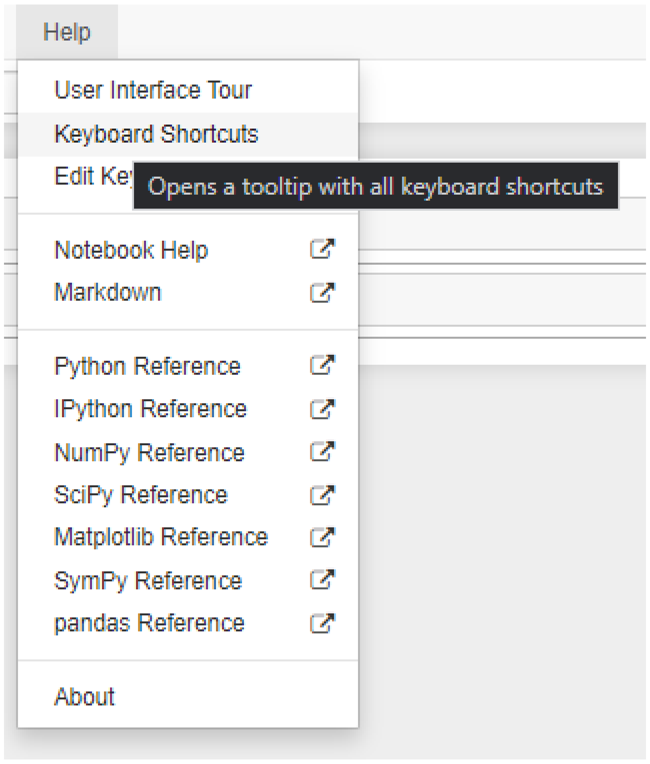
```
(base) C:\Users\shivaraj>jupyter notebook
[I 12:10:53.604 NotebookApp] JupyterLab extension loaded from C:\Users\shivaraj\Anaconda3\lib\site-package
s\jupyterlab
[I 12:10:53.604 NotebookApp] JupyterLab application directory is C:\Users\shivaraj\Anaconda3\share\jupyter
\lab
[I 12:10:53.604 NotebookApp] Serving notebooks from local directory: C:\Users\shivaraj
[I 12:10:53.604 NotebookApp] The Jupyter Notebook is running at:
[I 12:10:53.604 NotebookApp] http://localhost:8888/?token=8e071470bf6e0719331ef3d0cb3689f8d4465b22a37ff13a

[I 12:10:53.604 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip co
nfirmation).
[C 12:10:53.789 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/shivaraj/AppData/Roaming/jupyter/runtime/nbserver-1732-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=8e071470bf6e0719331ef3d0cb3689f8d4465b22a37ff13a
```

- Now start writing your first python program using Jupyter notebook, to open a new file go to your browser based interpreter.
- Click on New on top right side of the Jupyter notebook, select Python 3 and it will open another tab with new notebook.
- Now you can refer below images for your reference=

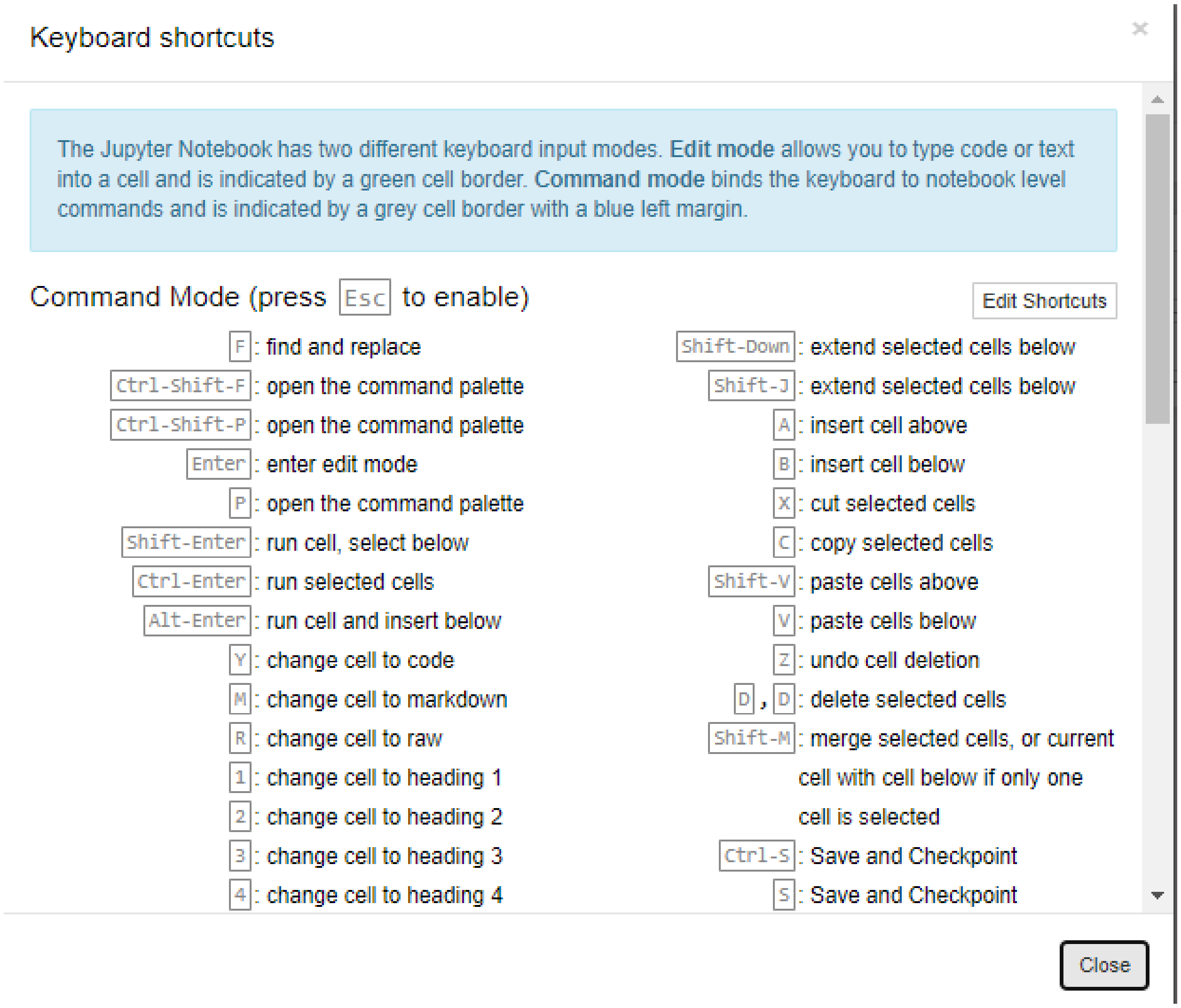




- To start our coding part you need to know some keyboard shortcuts to save, to run, go to next line and so on, for that go Help menu in Jupyter notebook and click on keyboard shortcuts.



- Now you will get list of different shortcut keys and you can remember common keys required for your program execution



Keyboard shortcuts

The Jupyter Notebook has two different keyboard input modes. **Edit mode** allows you to type code or text into a cell and is indicated by a green cell border. **Command mode** binds the keyboard to notebook level commands and is indicated by a grey cell border with a blue left margin.

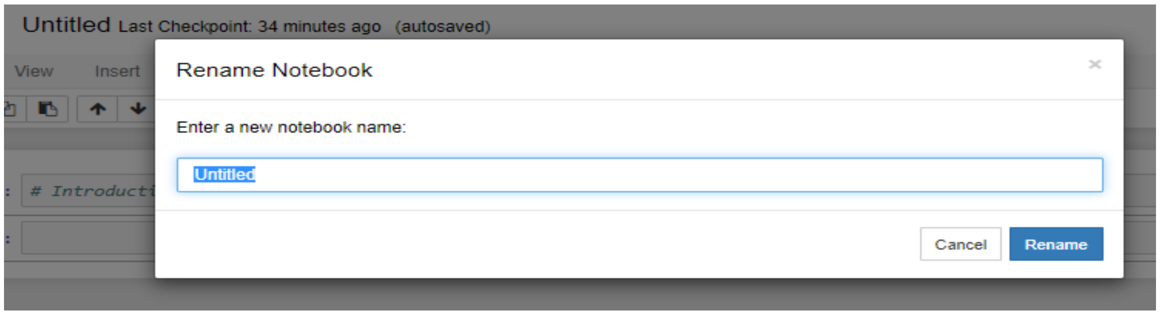
Command Mode (press `Esc` to enable)

Edit Shortcuts

`F`	find and replace	`Shift-Down`	extend selected cells below
`Ctrl-Shift-F`	open the command palette	`Shift-J`	extend selected cells below
`Ctrl-Shift-P`	open the command palette	`A`	insert cell above
`Enter`	enter edit mode	`B`	insert cell below
`P`	open the command palette	`X`	cut selected cells
`Shift-Enter`	run cell, select below	`C`	copy selected cells
`Ctrl-Enter`	run selected cells	`Shift-V`	paste cells above
`Alt-Enter`	run cell and insert below	`V`	paste cells below
`Y`	change cell to code	`Z`	undo cell deletion
`M`	change cell to markdown	`D`, `D`	delete selected cells
`R`	change cell to raw	`Shift-M`	merge selected cells, or current cell with cell below if only one cell is selected
`1`	change cell to heading 1	`Ctrl-S`	Save and Checkpoint
`2`	change cell to heading 2	`S`	Save and Checkpoint
`3`	change cell to heading 3		
`4`	change cell to heading 4		

- Once you are done with this click on close and start writing your first python program.

Go to your Jupyter Notebook, click on Untitled and rename the file as First Python Program(it can be your choice).

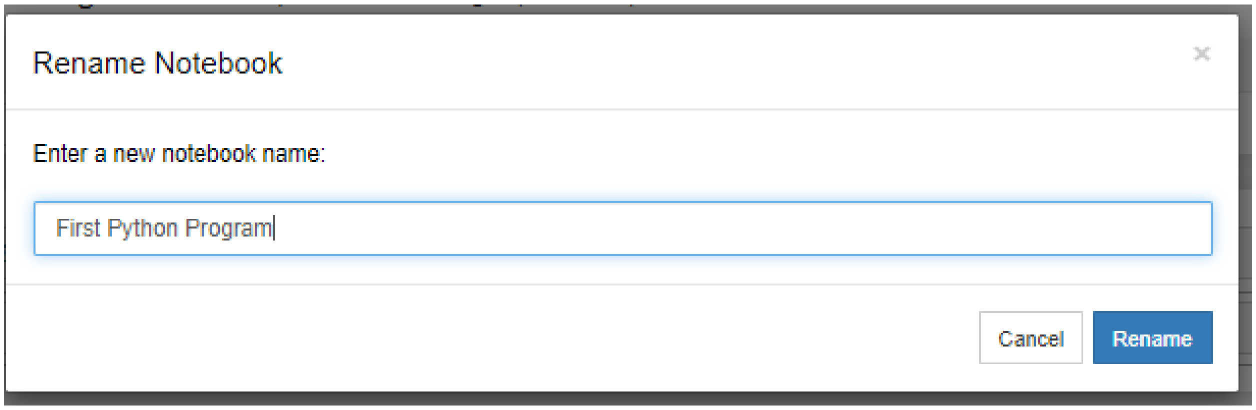


Untitled Last Checkpoint: 34 minutes ago (autosaved)

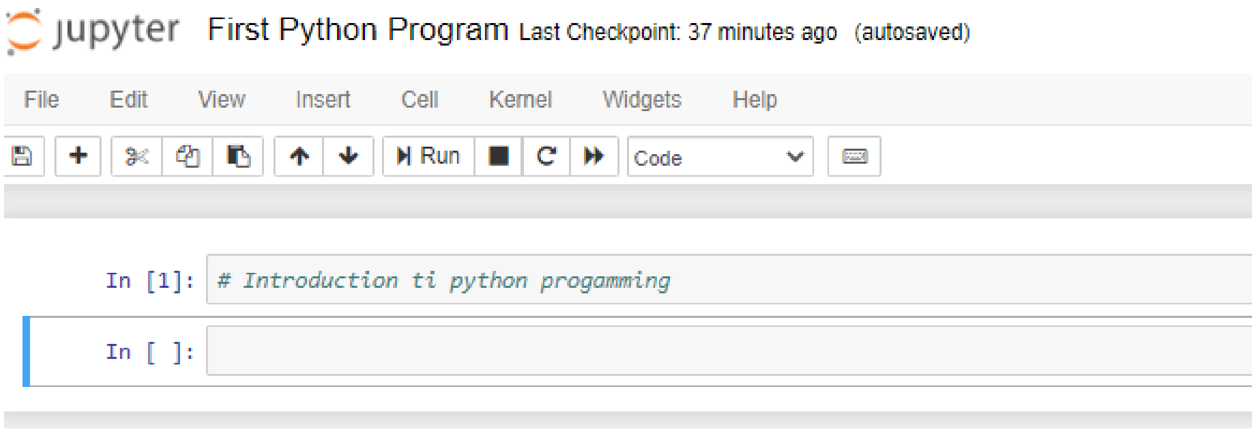
Rename Notebook

Enter a new notebook name:

Untitled



Now your file is renamed to First Python Program

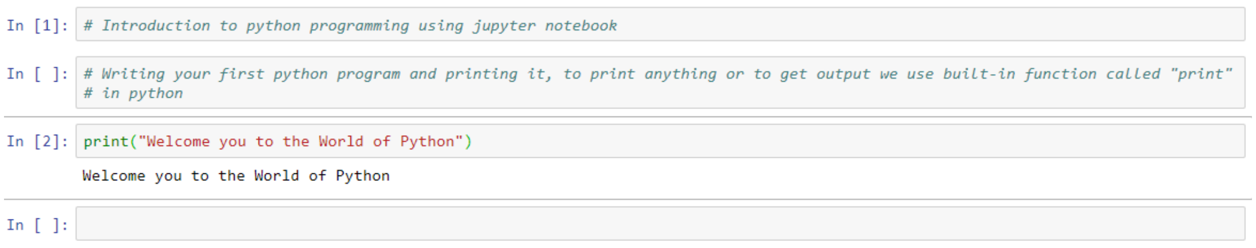

jupyter
First Python Program Last Checkpoint: 37 minutes ago (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

```
In [1]: # Introduction ti python progamming

In [ ]:
```

Some tips to write your python code:

- **#**: used to comment the line
- **Shift + Enter**: used to run the code
- **Enter**: used to go next line

From now onwards kindly follow the below images and simultaneously execute the code in Jupyter notebook.



```
In [1]: # Introduction to python programming using jupyter notebook

In [ ]: # Writing your first python program and printing it, to print anything or to get output we use built-in function called "print"
        # in python

In [2]: print("Welcome you to the World of Python")

        Welcome you to the World of Python

In [ ]:
```

Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming.

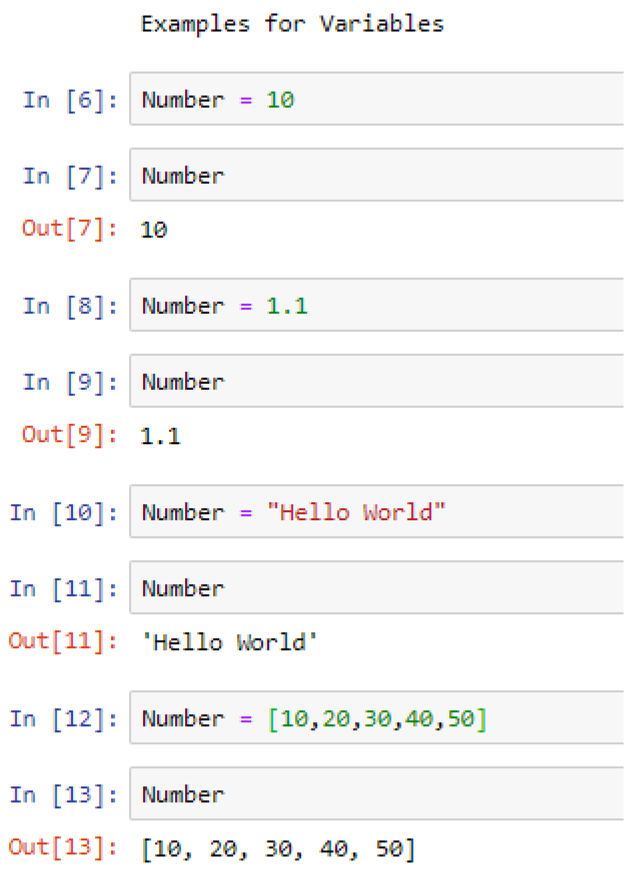
In clear: Data/Values can be stored in temporary storage spaces called variables

For example:

- number = 10
- number = 1.1

Initially, the value of number was 10. Later it's changed to 1.1(i.e.10 stored temporarily in variable called number and again 1.1 is stored in the same variable number)

In Python, we don't assign values to the variables, whereas Python gives the reference of the object (value) to the variable.

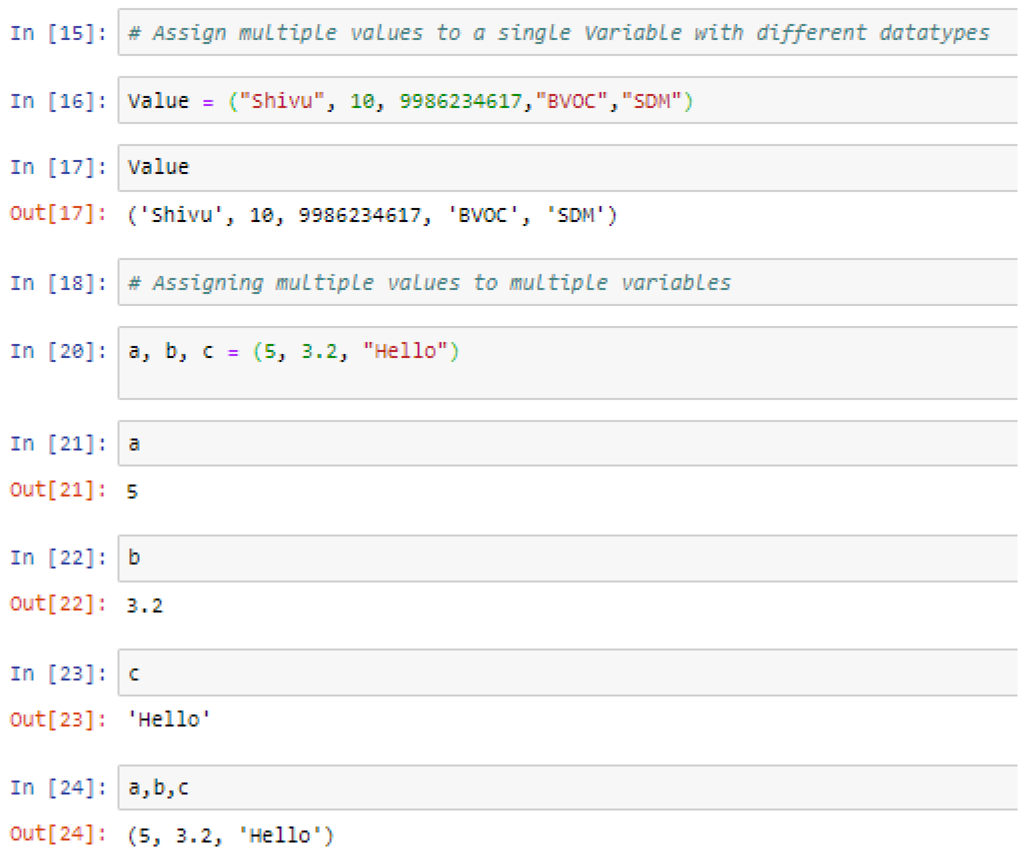


```
          Examples for Variables

 In [6]:  Number = 10

 In [7]:  Number
Out[7]:   10

 In [8]:  Number = 1.1

 In [9]:  Number
Out[9]:   1.1

In [10]:  Number = "Hello World"

In [11]:  Number
Out[11]:  'Hello World'

In [12]:  Number = [10,20,30,40,50]

In [13]:  Number
Out[13]:  [10, 20, 30, 40, 50]
```

- Now you can have a question how different values stored in a variable called Number, to achieve this we can use assignment operator called Equals (=).

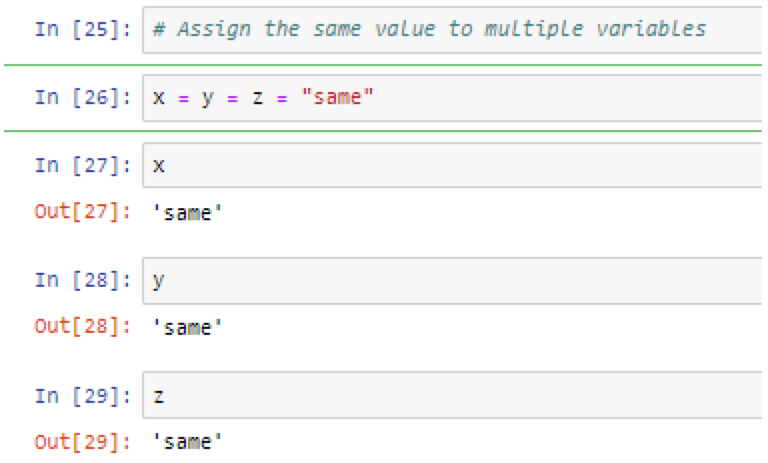
- So whatever the values like 10, 1.1, "Hello World", [10, 20, 30, 40, 50] assigned to a variable called Number.
- Every time we can change the variable value with other values and that is called assigning a new value to a variable called Number.

We can also perform below operations:



```
In [15]: # Assign multiple values to a single Variable with different datatypes

In [16]: Value = ("Shivu", 10, 9986234617,"BVOC","SDM")

In [17]: Value
Out[17]: ('Shivu', 10, 9986234617, 'BVOC', 'SDM')

In [18]: # Assigning multiple values to multiple variables

In [20]: a, b, c = (5, 3.2, "Hello")

In [21]: a
Out[21]: 5

In [22]: b
Out[22]: 3.2

In [23]: c
Out[23]: 'Hello'

In [24]: a,b,c
Out[24]: (5, 3.2, 'Hello')
```

Assign the same value to multiple variables

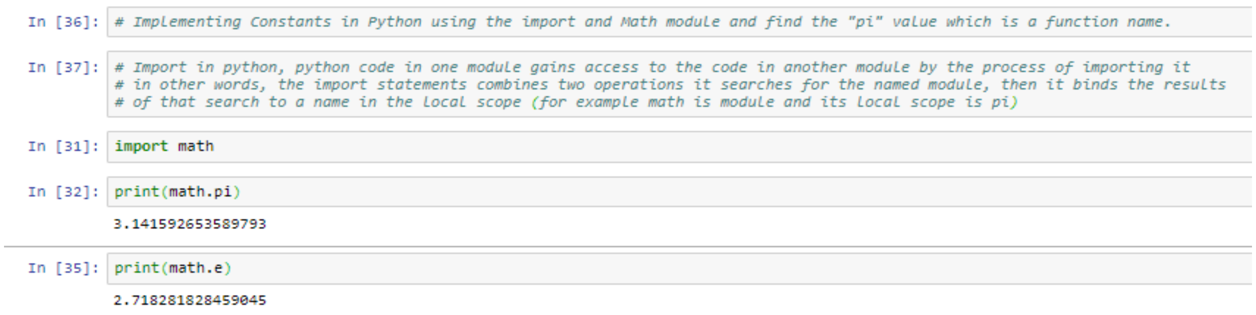


```
In [25]: # Assign the same value to multiple variables

In [26]: x = y = z = "same"

In [27]: x
Out[27]: 'same'

In [28]: y
Out[28]: 'same'

In [29]: z
Out[29]: 'same'
```

Constants

- A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.
- **Assigning value to a constant in Python**: constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc. which is **imported** (import function) to main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example:



```
In [36]: # Implementing Constants in Python using the import and Math module and find the "pi" value which is a function name.

In [37]: # Import in python, python code in one module gains access to the code in another module by the process of importing it
         # in other words, the import statements combines two operations it searches for the named module, then it binds the results
         # of that search to a name in the Local scope (for example math is module and its Local scope is pi)

In [31]: import math

In [32]: print(math.pi)

         3.141592653589793

In [35]: print(math.e)

         2.718281828459045
```

Rules and Naming Convention for Variables and constants

Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). For example:

For example:

- ❖ snake_case
- ❖ MACRO_CASE
- ❖ camelCase
- ❖ CapWords

Create a name that makes sense. For example, vowel makes more sense than v (single letter).

If you want to create a variable name having two words, use underscore to separate them.

For example:

- my_name
- current_salary
- Use capital letters possible to declare a constant. For example:
- PI, G, MASS, SPEED_OF_LIGHT, TEMP

Never use special symbols like !, @, #, $, %, etc.

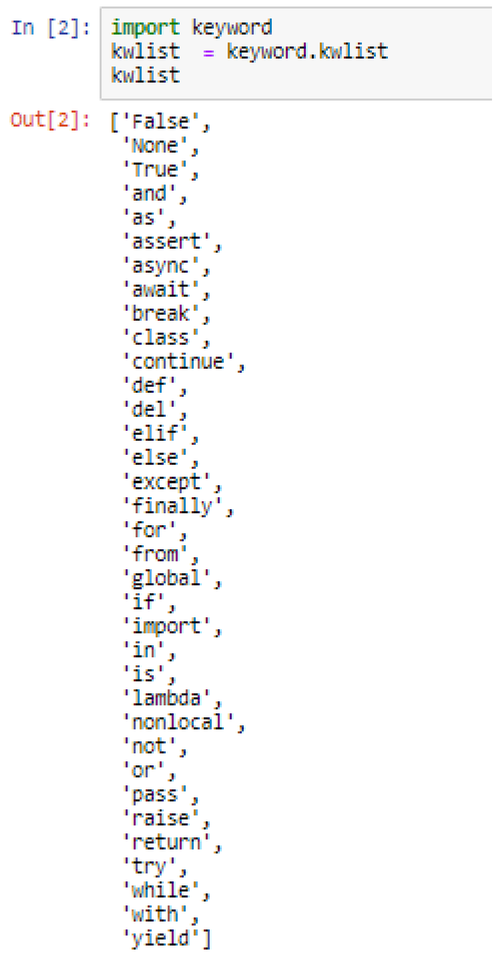
Finally, don’t start a variable name with a digit.

Python Keywords and Identifiers

- In Python keyword is a unique programming term intended to perform some action. There are as many as 33 such keywords in Python, each serving a different purpose. Together, they build the vocabulary of the Python language.
- They represent the syntax and structure of a Python program. Since all of them are reserved, so you can't use their names for defining variables, classes or functions.
- Keywords are special words which are reserved and have a specific meaning. Python has a set of keywords that cannot be used as variables in program.
- All keywords in Python are case sensitive. So, you must be careful while using them in your code. We've just captured here a snapshot of the possible Python keywords.

Python Keywords

- The keyword module in Python's standard library allows a Python program to determine if a string is a keyword.
- **keyword.kwlist:** This attribute returns Sequence containing all the keywords defined for the interpreter.



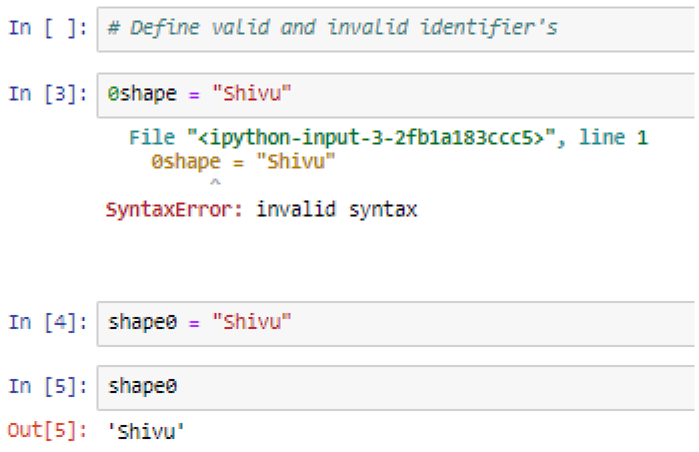
```
In [2]: import keyword
        kwlist  = keyword.kwlist
        kwlist

Out[2]: ['False',
         'None',
         'True',
         'and',
         'as',
         'assert',
         'async',
         'await',
         'break',
         'class',
         'continue',
         'def',
         'del',
         'elif',
         'else',
         'except',
         'finally',
         'for',
         'from',
         'global',
         'if',
         'import',
         'in',
         'is',
         'lambda',
         'nonlocal',
         'not',
         'or',
         'pass',
         'raise',
         'return',
         'try',
         'while',
         'with',
         'yield']
```

Python Identifiers

- Python Identifiers are user-defined names to represent a variable, function, class, module or any other object. If you assign some name to a programmable entity in Python, then it is nothing but technically called an identifier.
- Python language lays down a set of rules for programmers to create meaningful identifiers.

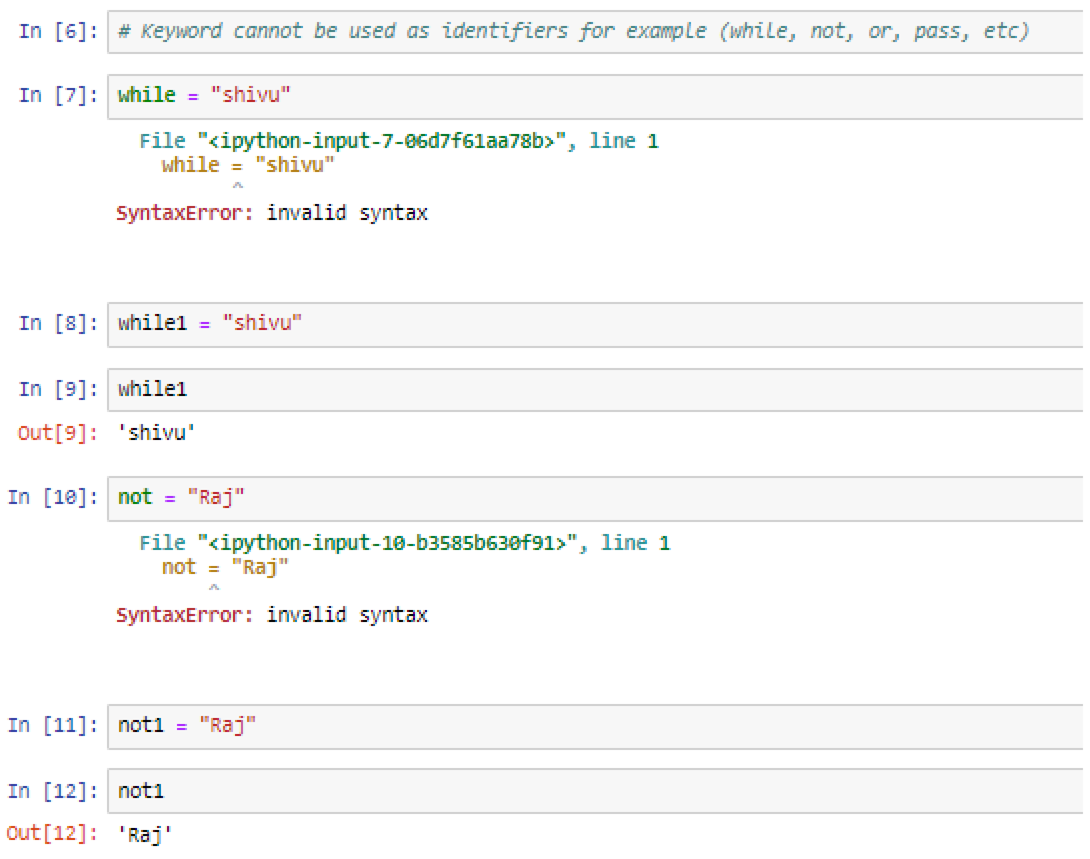
Guidelines for Creating Identifiers in Python

- To form an identifier, use a **sequence of letters** either in **lowercase (a to z)** or **uppercase (A to Z)**. However, you can also mix up **digits (0 to 9)** or an **underscore (_)** while writing an identifier.
- You can't use digits to begin an identifier name. It'll lead to the syntax error.
  For example – The name, **0Shape** is incorrect, but **shape0** is a valid identifier.



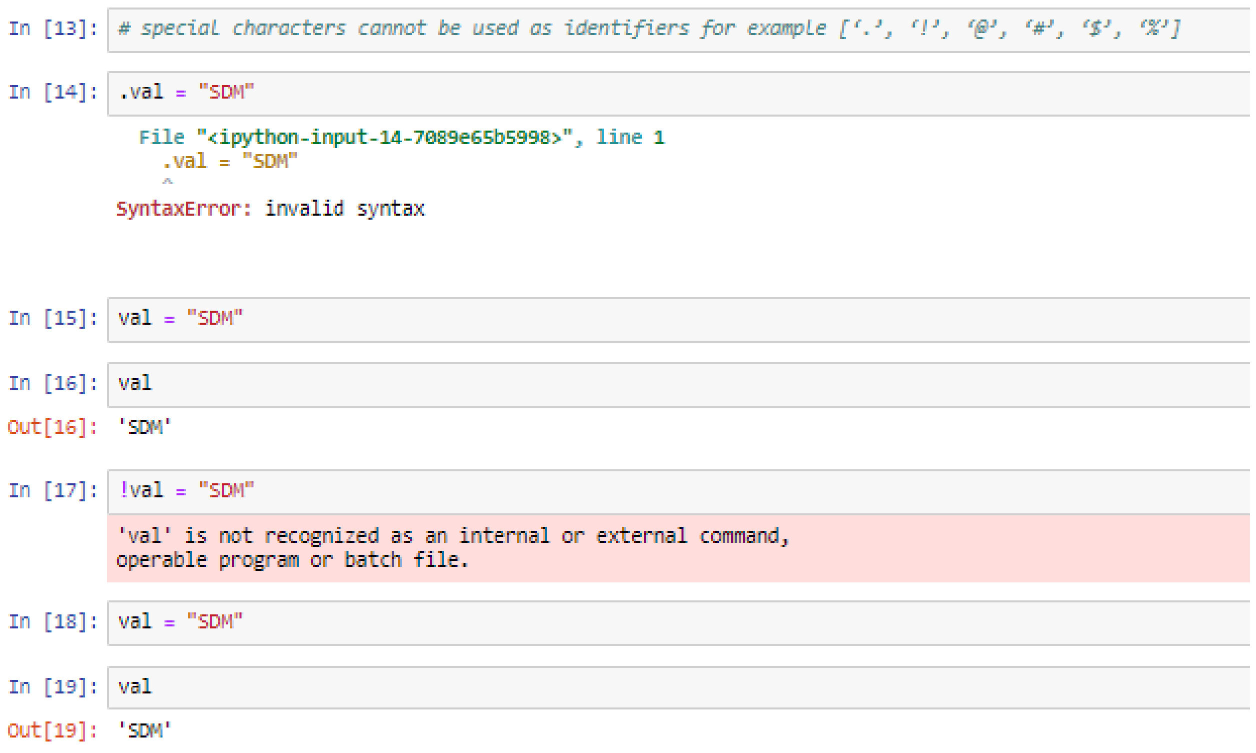
```
In [ ]: # Define valid and invalid identifier's

In [3]: 0shape = "Shivu"

  File "<ipython-input-3-2fb1a183ccc5>", line 1
    0shape = "Shivu"
         ^
SyntaxError: invalid syntax


In [4]: shape0 = "Shivu"

In [5]: shape0
Out[5]: 'Shivu'
```

- Also, the Keywords are reserved, so you cannot use them as identifiers. Similarly Python Identifiers can also not have special characters **['.', '!', '@', '#', '$', '%']** in their formation.

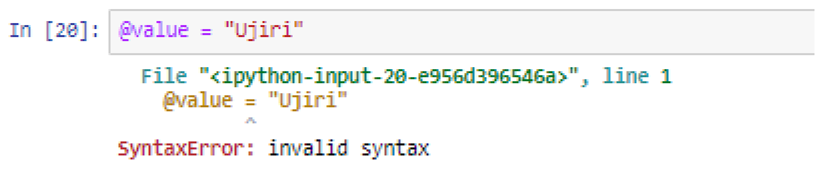


```
In [6]: # Keyword cannot be used as identifiers for example (while, not, or, pass, etc)

In [7]: while = "shivu"

          File "<ipython-input-7-06d7f61aa78b>", line 1
            while = "shivu"
                  ^
        SyntaxError: invalid syntax


In [8]: while1 = "shivu"

In [9]: while1

Out[9]: 'shivu'

In [10]: not = "Raj"

          File "<ipython-input-10-b3585b630f91>", line 1
            not = "Raj"
                ^
        SyntaxError: invalid syntax


In [11]: not1 = "Raj"

In [12]: not1

Out[12]: 'Raj'
```

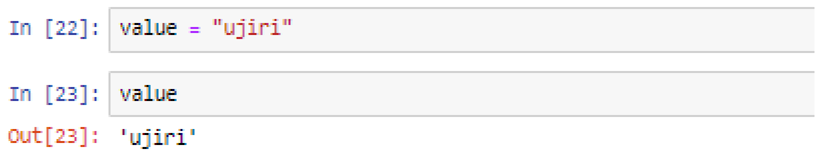
Examples for Special characters:



```
In [13]: # special characters cannot be used as identifiers for example ['.', '!', '@', '#', '$', '%']

In [14]: .val = "SDM"

          File "<ipython-input-14-7089e65b5998>", line 1
            .val = "SDM"
            ^
        SyntaxError: invalid syntax


In [15]: val = "SDM"

In [16]: val

Out[16]: 'SDM'

In [17]: !val = "SDM"

         'val' is not recognized as an internal or external command,
         operable program or batch file.

In [18]: val = "SDM"

In [19]: val

Out[19]: 'SDM'
```



```
In [20]: @value = "Ujiri"

  File "<ipython-input-20-e956d396546a>", line 1
    @value = "Ujiri"
          ^
SyntaxError: invalid syntax
```



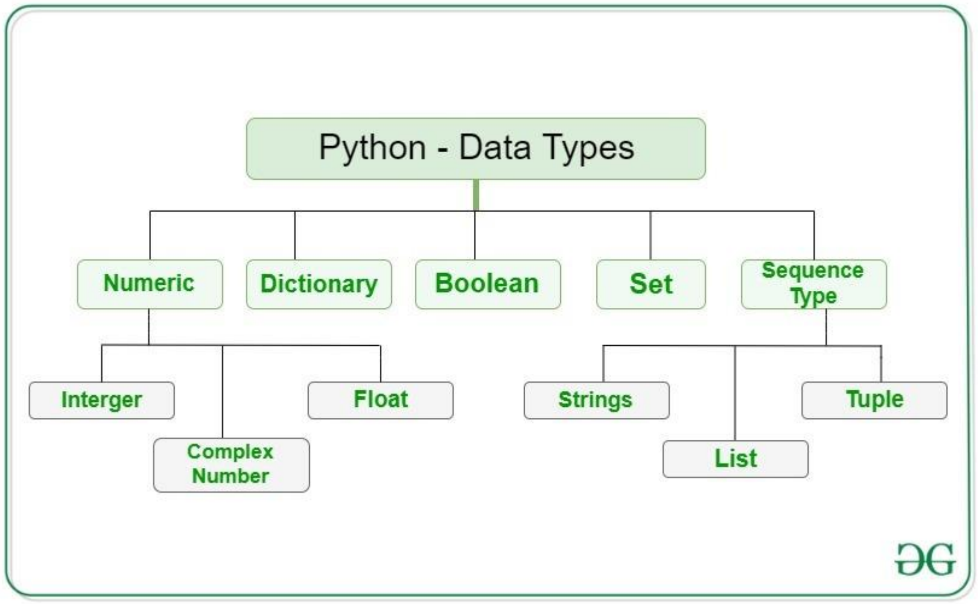
```
In [22]: value = "ujiri"

In [23]: value
Out[23]: 'ujiri'
```

Data types

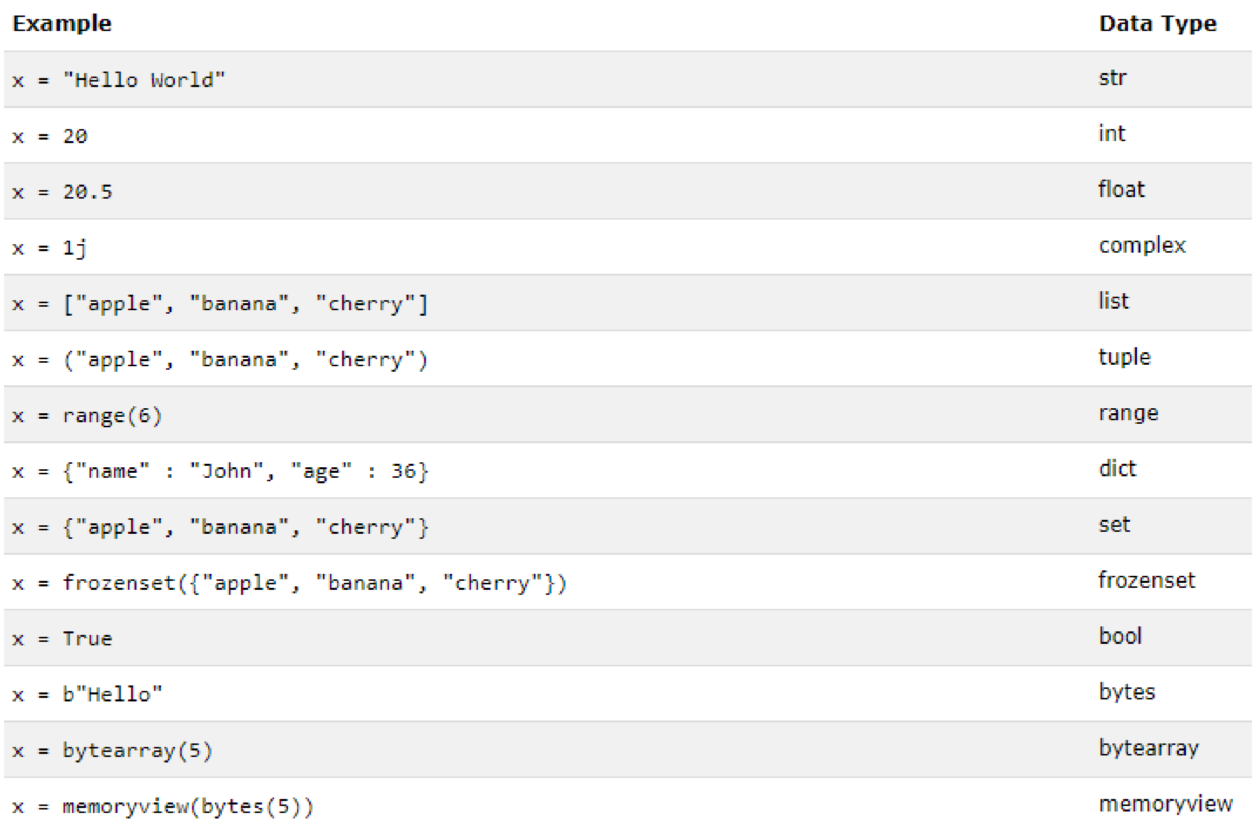
In programming, data type is an important concept and Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: list, tuple, range
- Mapping Type: dict
- Set Types: set, frozenset
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview

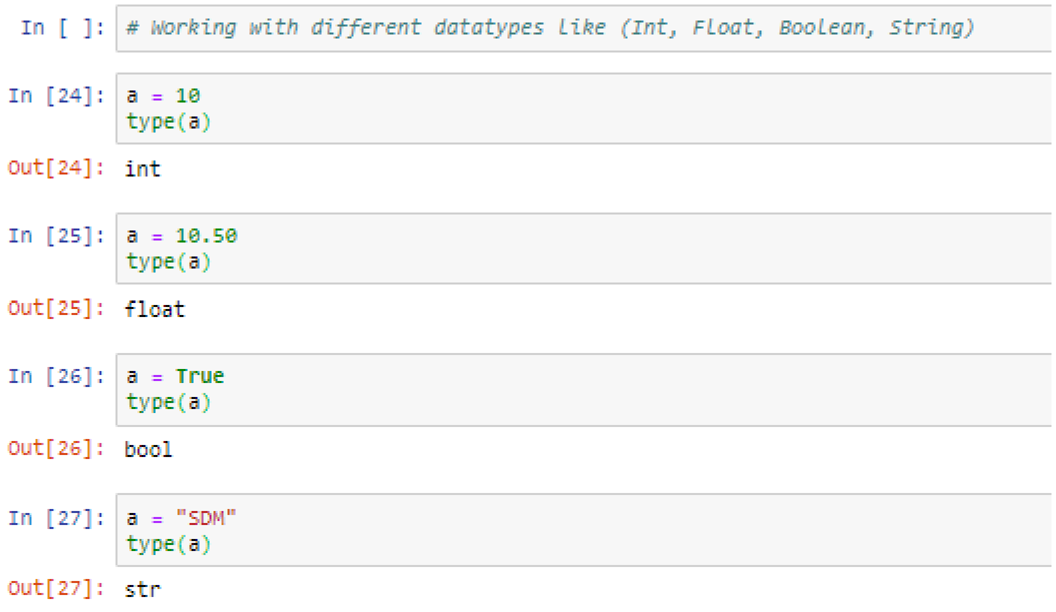


Setting the Data Type: In Python, the data type is set when you assign a value to a variable



Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Some of the simple example for datatypes in python

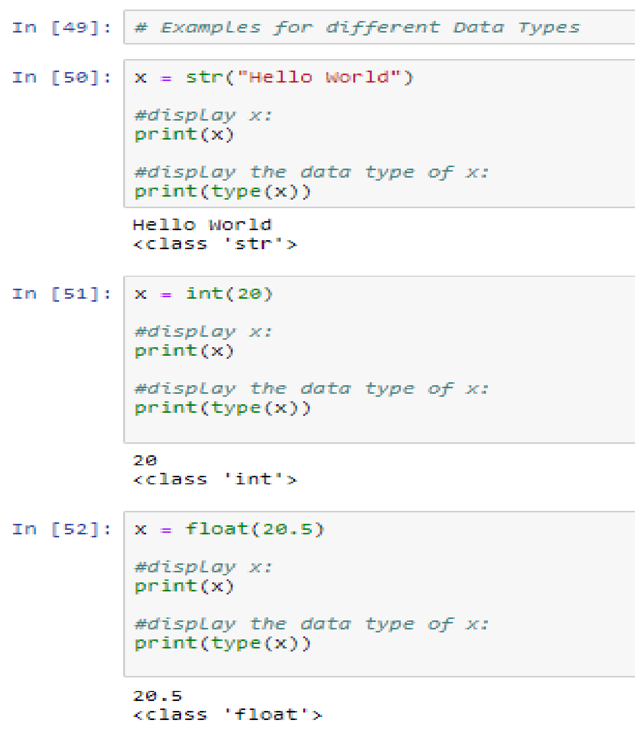


```
In [ ]:  # Working with different datatypes like (Int, Float, Boolean, String)

In [24]: a = 10
         type(a)

Out[24]: int

In [25]: a = 10.50
         type(a)

Out[25]: float

In [26]: a = True
         type(a)

Out[26]: bool

In [27]: a = "SDM"
         type(a)

Out[27]: str
```

Setting the Specific Data Type: If you want to specify the data type, you can use the following constructor functions

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Below syntax will show you the how datatypes are working and also define to which class they belongs to.



```
In [49]: # Examples for different Data Types
```

```
In [50]: x = str("Hello World")

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

         Hello World
         <class 'str'>
```

```
In [51]: x = int(20)

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

         20
         <class 'int'>
```

```
In [52]: x = float(20.5)

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

         20.5
         <class 'float'>
```

In [53]:
```python
x = complex(1j)

#display x:
print(x)

#display the data type of x:
print(type(x))
```
```
1j
<class 'complex'>
```

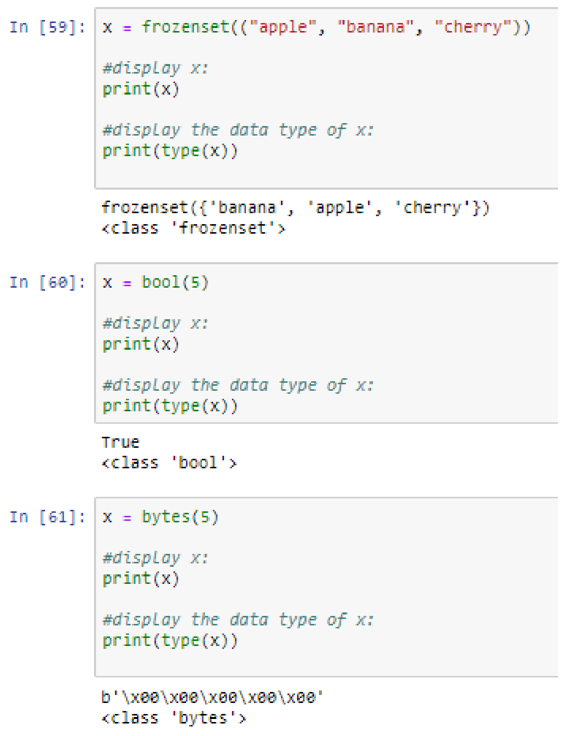
In [54]:
```python
x = list(("apple", "banana", "cherry"))

#display x:
print(x)

#display the data type of x:
print(type(x))
```
```
['apple', 'banana', 'cherry']
<class 'list'>
```

In [55]:
```python
x = tuple(("apple", "banana", "cherry"))

#display x:
print(x)

#display the data type of x:
print(type(x))
```
```
('apple', 'banana', 'cherry')
<class 'tuple'>
```

In [56]:
```python
x = range(6)

#display x:
print(x)

#display the data type of x:
print(type(x))
```
```
range(0, 6)
<class 'range'>
```

In [57]:
```python
x = dict(name="John", age=36)

#display x:
print(x)

#display the data type of x:
print(type(x))
```
```
{'name': 'John', 'age': 36}
<class 'dict'>
```

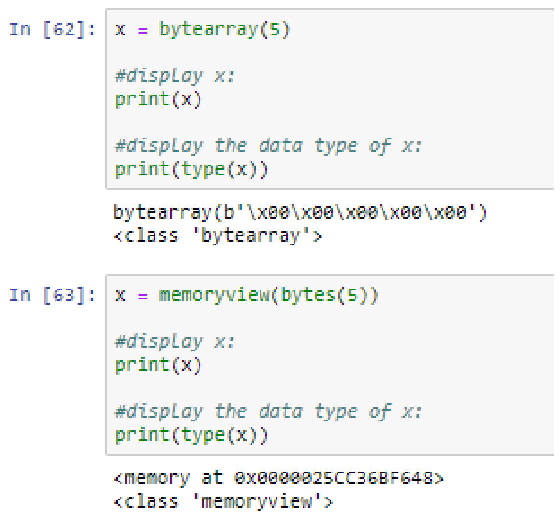
In [58]:
```python
x = set(("apple", "banana", "cherry"))

#display x:
print(x)

#display the data type of x:
print(type(x))
```
```
{'banana', 'apple', 'cherry'}
<class 'set'>
```



```
In [59]: x = frozenset(("apple", "banana", "cherry"))

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

frozenset({'banana', 'apple', 'cherry'})
<class 'frozenset'>
```

```
In [60]: x = bool(5)

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

True
<class 'bool'>
```

```
In [61]: x = bytes(5)

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

b'\x00\x00\x00\x00\x00'
<class 'bytes'>
```



```
In [62]: x = bytearray(5)

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

bytearray(b'\x00\x00\x00\x00\x00')
<class 'bytearray'>
```

```
In [63]: x = memoryview(bytes(5))

         #display x:
         print(x)

         #display the data type of x:
         print(type(x))

<memory at 0x0000025CC36BF648>
<class 'memoryview'>
```

The mutability of an object is determined by its type

- Some objects contain references to other objects, these objects are called **containers**.
- Some examples of containers are a **tuple**, **list**, and **dictionary**.
- The **value of an immutable container** that contains a **reference to a mutable object can be changed** if that mutable object is changed

Mutable and Immutable Data Types in Python

- Some of the **mutable** data types in Python are **list, dictionary, set** and **user-defined classes**.
- On the other hand, some of the **immutable** data types are **int, float, decimal, bool, string, tuple, and range**.
- Every value in Python has a data type. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes

Python Operators

- Operators are used to perform operations on variables and values.
- Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations

Operator	**Name**	**Example**
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

```
In [66]: # Python Arithmetic Operators
```

```
In [67]: x = 5
         y = 3

         print(x + y)

         8
```

```
In [68]: x = 5
         y = 3

         print(x - y)

         2
```

```
In [69]: x = 5
         y = 3

         print(x * y)

         15
```

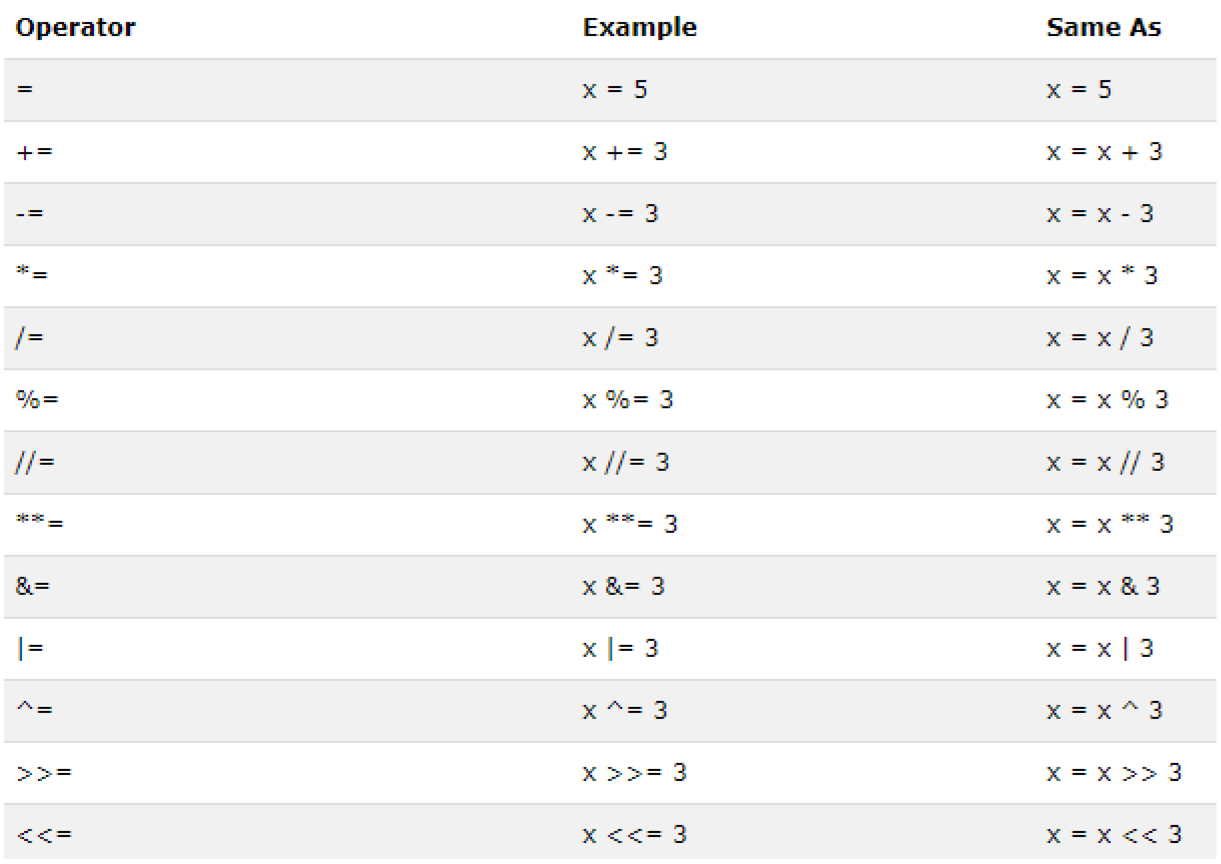
```
In [71]: x = 12
         y = 3

         print(x / y)

         4.0
```

```
In [72]: x = 5
         y = 2

         print(x % y)

         1
```

```
In [73]: x = 2
         y = 5

         print(x ** y) #same as 2*2*2*2*2

         32
```

```
In [74]: x = 15
         y = 2

         print(x // y)

         #the floor division // rounds the result down to the nearest whole number

         7
```

Python Assignment Operators: Assignment operators are used to assign values to variables



Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

=: Assigns values from right side operands to left side operand

+=: Add AND, It adds right operand to the left operand and assign the result to left operand

-=: Subtract AND, It subtracts right operand from the left operand and assign the result to left operand

*=: Multiply AND, It multiplies right operand with the left operand and assign the result to left operand

/=: Divide AND, It divides left operand with the right operand and assign the result to left operand

%=: Modulus AND, It takes modulus using two operands and assign the result to left operand

//=: Floor Division, It performs floor division on operators and assign value to the left operand

**=: Exponent AND, Performs exponential (power) calculation on operators and assign value to the left operand

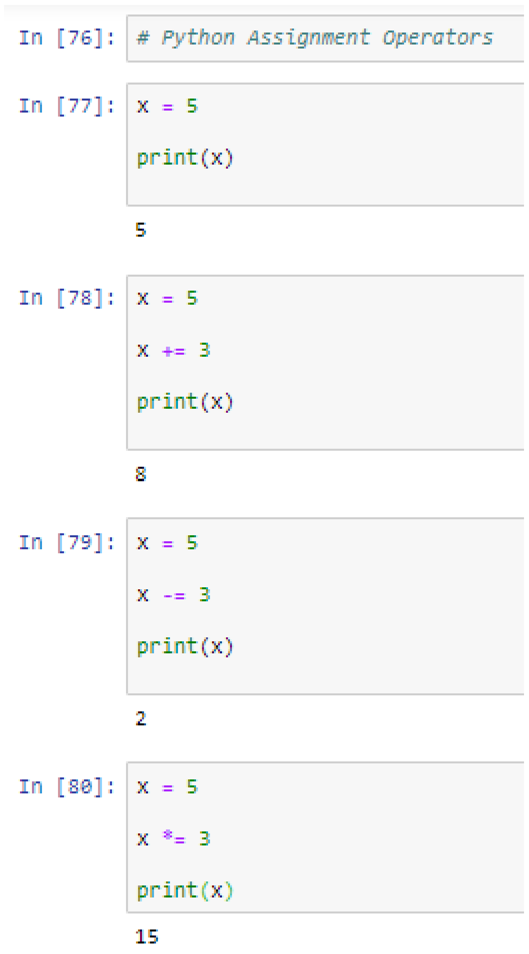
&=: Performs Bitwise AND on operands and assign value to left operand

|=: Performs Bitwise OR on operands and assign value to left operand

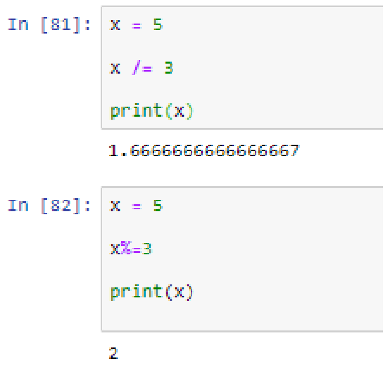
^=: Performs Bitwise XOR on operands and assign value to left operand

>>=: Performs Bitwise right shift on operands and assign value to left operand

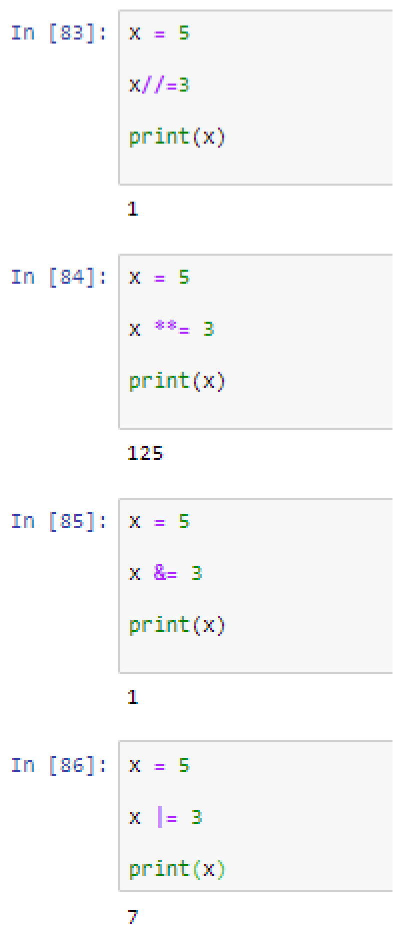
<<=: Performs Bitwise left shift on operands and assign value to left operand



```
In [76]: # Python Assignment Operators

In [77]: x = 5

         print(x)

5

In [78]: x = 5

         x += 3

         print(x)

8

In [79]: x = 5

         x -= 3

         print(x)

2

In [80]: x = 5

         x *= 3

         print(x)

15
```



```
In [81]: x = 5

         x /= 3

         print(x)

1.6666666666666667

In [82]: x = 5

         x%=3

         print(x)

2
```

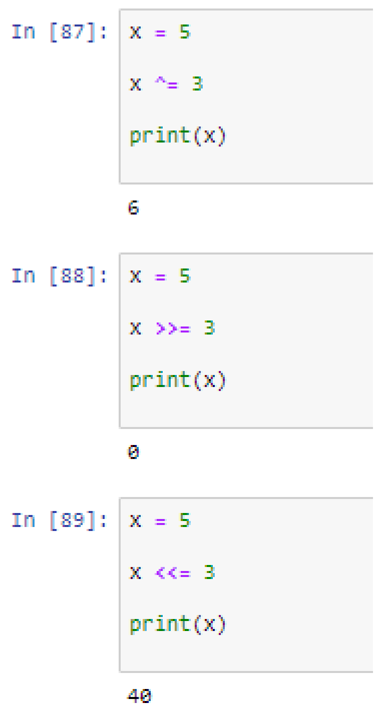


```
In [83]: x = 5

         x//=3

         print(x)

1
```

```
In [84]: x = 5

         x **= 3

         print(x)

125
```

```
In [85]: x = 5

         x &= 3

         print(x)

1
```

```
In [86]: x = 5

         x |= 3

         print(x)

7
```

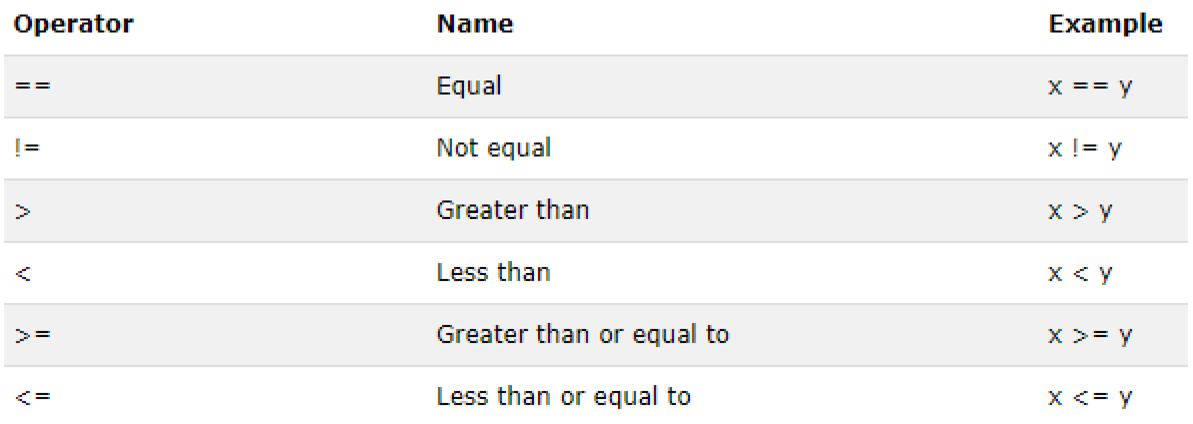


```
In [87]: x = 5

         x ^= 3

         print(x)

6
```

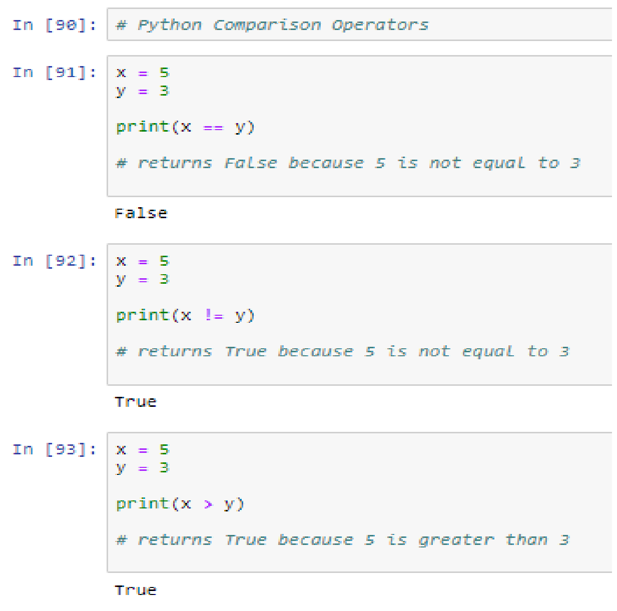
```
In [88]: x = 5

         x >>= 3

         print(x)

0
```

```
In [89]: x = 5

         x <<= 3

         print(x)

40
```

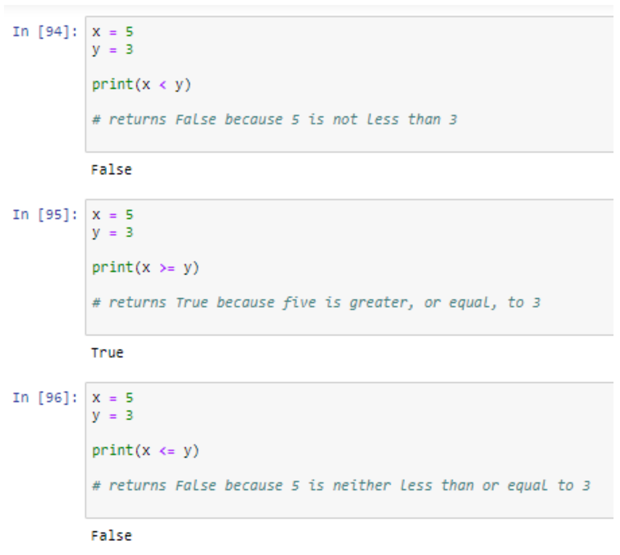
Python Comparison Operators: Comparison operators are used to compare two values



Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y



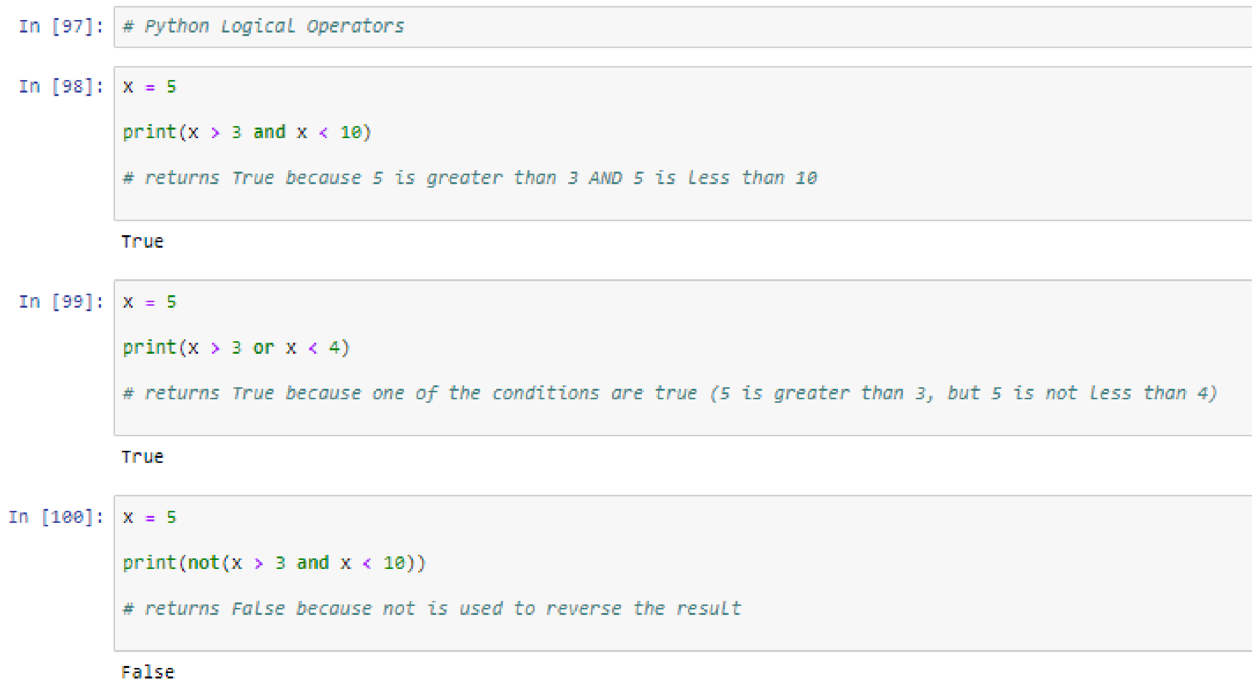
```
In [90]: # Python Comparison Operators

In [91]: x = 5
         y = 3

         print(x == y)

         # returns False because 5 is not equal to 3

         False

In [92]: x = 5
         y = 3

         print(x != y)

         # returns True because 5 is not equal to 3

         True

In [93]: x = 5
         y = 3

         print(x > y)

         # returns True because 5 is greater than 3

         True
```



```
In [94]: x = 5
         y = 3

         print(x < y)

         # returns False because 5 is not less than 3

         False

In [95]: x = 5
         y = 3

         print(x >= y)

         # returns True because five is greater, or equal, to 3

         True

In [96]: x = 5
         y = 3

         print(x <= y)

         # returns False because 5 is neither less than or equal to 3

         False
```

Python Logical Operators: Logical operators are used to combine conditional statements

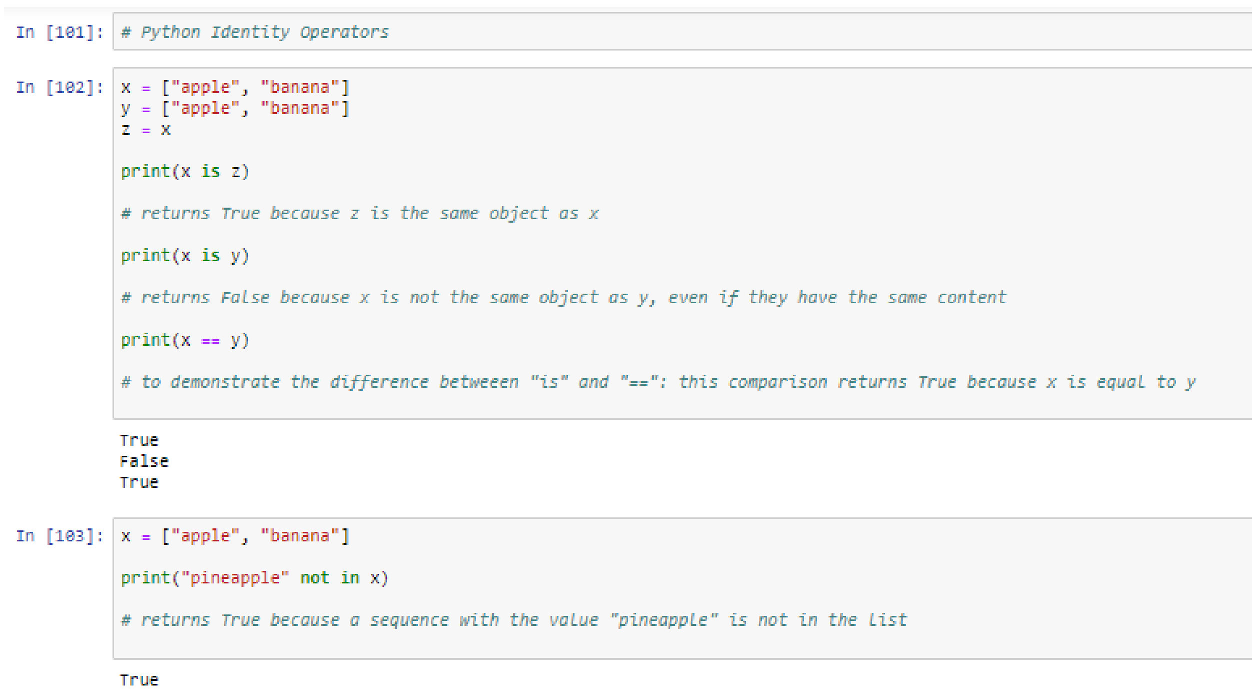
Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)



```
In [97]: # Python Logical Operators

In [98]: x = 5

         print(x > 3 and x < 10)

         # returns True because 5 is greater than 3 AND 5 is less than 10

         True

In [99]: x = 5

         print(x > 3 or x < 4)

         # returns True because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)

         True

In [100]: x = 5

          print(not(x > 3 and x < 10))

          # returns False because not is used to reverse the result

          False
```

Python Identity Operators: Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y



```
In [101]: # Python Identity Operators
```

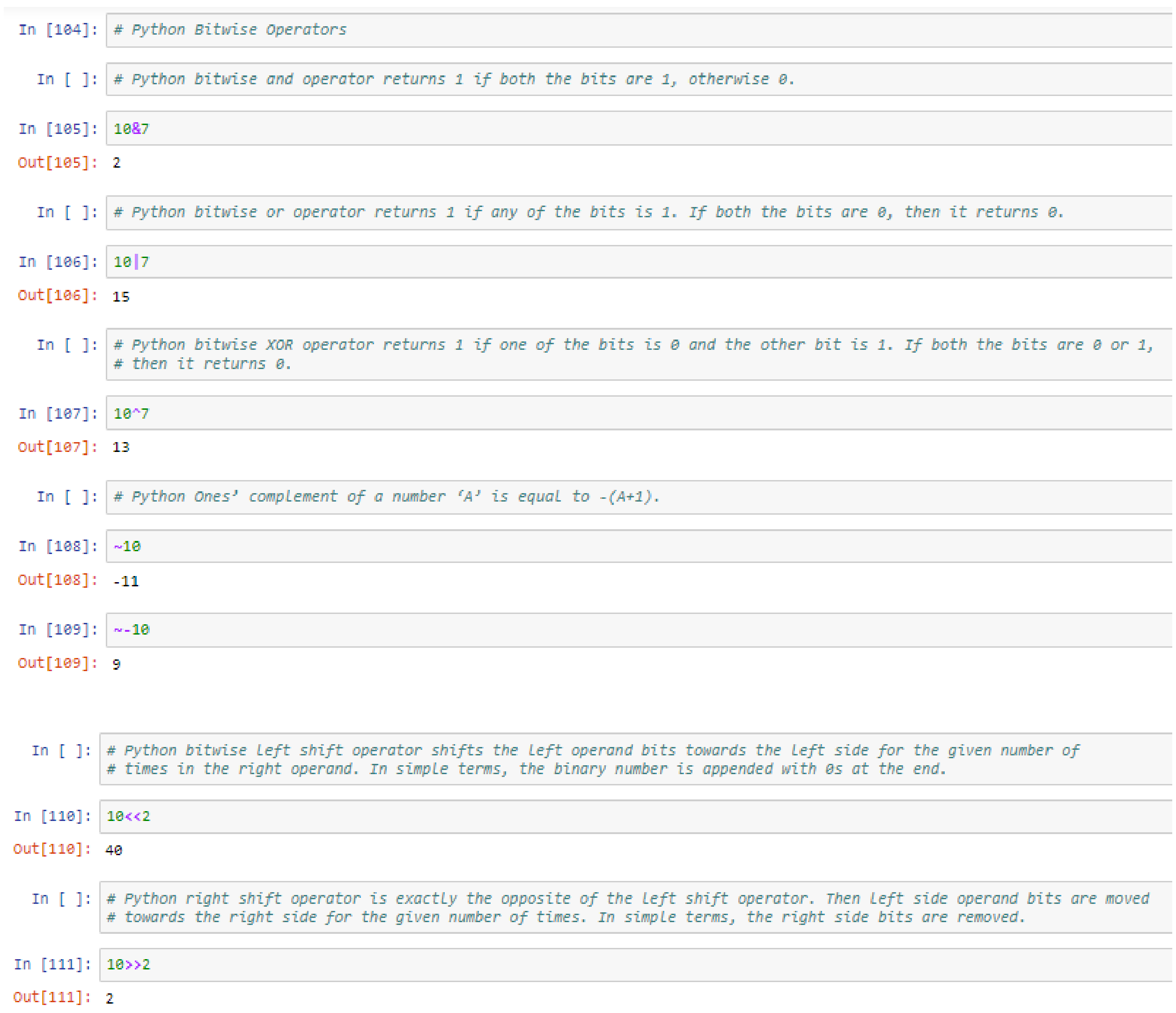
```
In [102]: x = ["apple", "banana"]
          y = ["apple", "banana"]
          z = x

          print(x is z)

          # returns True because z is the same object as x

          print(x is y)

          # returns False because x is not the same object as y, even if they have the same content

          print(x == y)

          # to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y

          True
          False
          True
```

```
In [103]: x = ["apple", "banana"]

          print("pineapple" not in x)

          # returns True because a sequence with the value "pineapple" is not in the list

          True
```

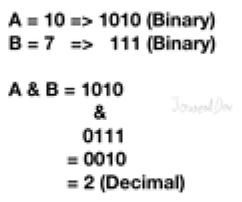
Python bitwise operators

- Python bitwise operators are used to perform bitwise calculations on integers. The integers are converted into binary format and then operations are performed bit by bit, hence the name bitwise operators.
- Python bitwise operators work on integers only and the final output is returned in the decimal format. Python bitwise operators are also called binary operators.

Bitwise Operator	Description	Simple Example
&	Bitwise AND Operator	10 & 7 = 2
|	Bitwise OR Operator	10 | 7 = 15
^	Bitwise XOR Operator	10 ^ 7 = 13
~	Bitwise Ones' Compliment Operator	~10 = -11
<<	Bitwise Left Shift operator	10<<2 = 40
>>	Bitwise Right Shift Operator	10>>1 = 5

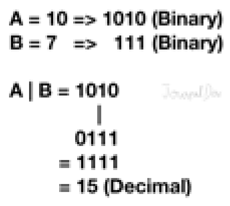


```
In [104]: # Python Bitwise Operators

In [ ]: # Python bitwise and operator returns 1 if both the bits are 1, otherwise 0.

In [105]: 10&7
Out[105]: 2

In [ ]: # Python bitwise or operator returns 1 if any of the bits is 1. If both the bits are 0, then it returns 0.

In [106]: 10|7
Out[106]: 15

In [ ]: # Python bitwise XOR operator returns 1 if one of the bits is 0 and the other bit is 1. If both the bits are 0 or 1,
        # then it returns 0.

In [107]: 10^7
Out[107]: 13

In [ ]: # Python Ones’ complement of a number ‘A’ is equal to -(A+1).

In [108]: ~10
Out[108]: -11

In [109]: ~-10
Out[109]: 9


In [ ]: # Python bitwise left shift operator shifts the left operand bits towards the left side for the given number of
        # times in the right operand. In simple terms, the binary number is appended with 0s at the end.

In [110]: 10<<2
Out[110]: 40

In [ ]: # Python right shift operator is exactly the opposite of the left shift operator. Then left side operand bits are moved
        # towards the right side for the given number of times. In simple terms, the right side bits are removed.

In [111]: 10>>2
Out[111]: 2
```



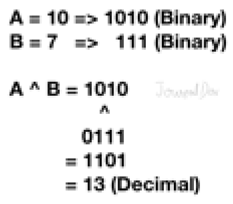
```
A = 10 => 1010 (Binary)
B = 7  =>  111 (Binary)

A & B = 1010
          &
         0111
       = 0010
       = 2 (Decimal)
```

Bitwise AND Operator



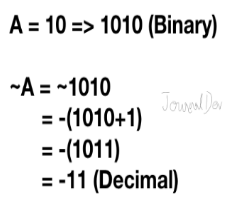
```
A = 10 => 1010 (Binary)
B = 7  =>  111 (Binary)

A | B = 1010
          |
         0111
       = 1111
       = 15 (Decimal)
```

Bitwise OR Operator



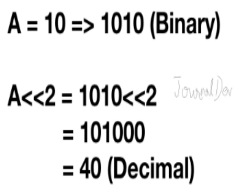
```
A = 10 => 1010 (Binary)
B = 7  =>  111 (Binary)

A ^ B = 1010
          ^
         0111
       = 1101
       = 13 (Decimal)
```

Bitwise XOR Operator



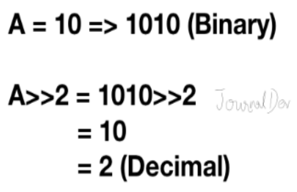
```
A = 10 => 1010 (Binary)

~A = ~1010
   = -(1010+1)
   = -(1011)
   = -11 (Decimal)
```

Bitwise Ones’ Complement Operator



```
A = 10 => 1010 (Binary)

A<<2 = 1010<<2
     = 101000
     = 40 (Decimal)
```

Bitwise Left Shift Operator



```
A = 10 => 1010 (Binary)

A>>2 = 1010>>2
     = 10
     = 2 (Decimal)
```

Bitwise Right Shift Operator

Python Strings:

- Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable.

For example

var1 = 'Hello World!'

var2 = "Python Programming"

Triple Quotes: Python's triple quotes comes to the rescue by allowing strings to span multiple lines

var3 = """this is a long string that is made up of

Several lines and non-printable characters such as

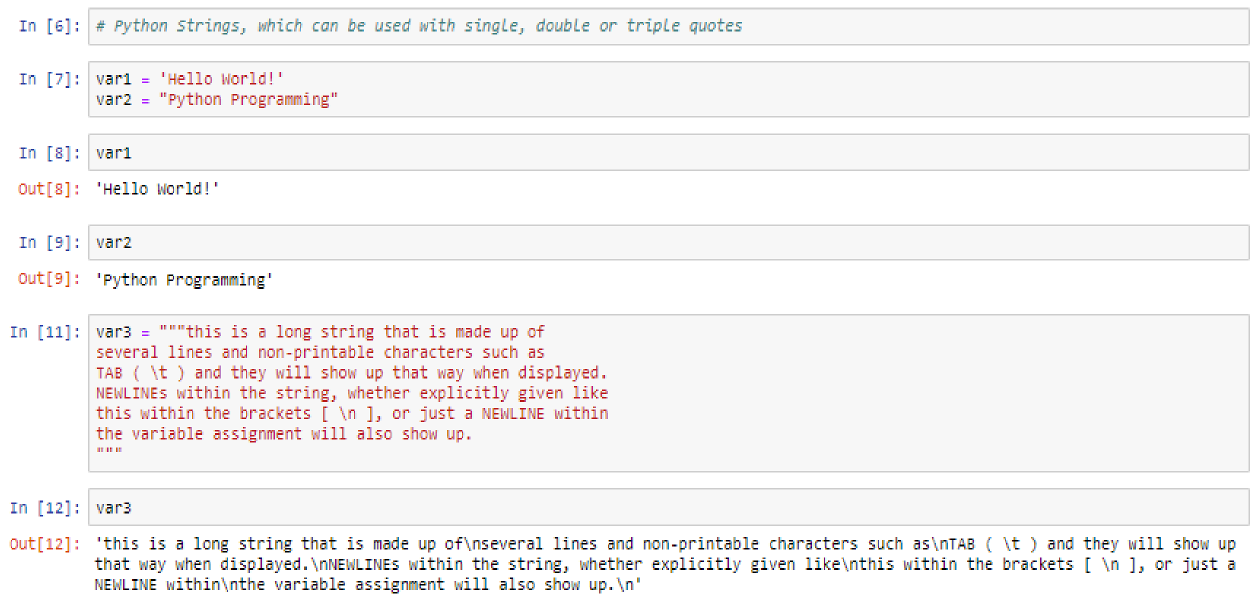
TAB (\t) and they will show up that way when displayed.

NEWLINEs within the string, whether explicitly given like

This within the brackets [\n], or just a NEWLINE within

The variable assignment will also show up.

"""

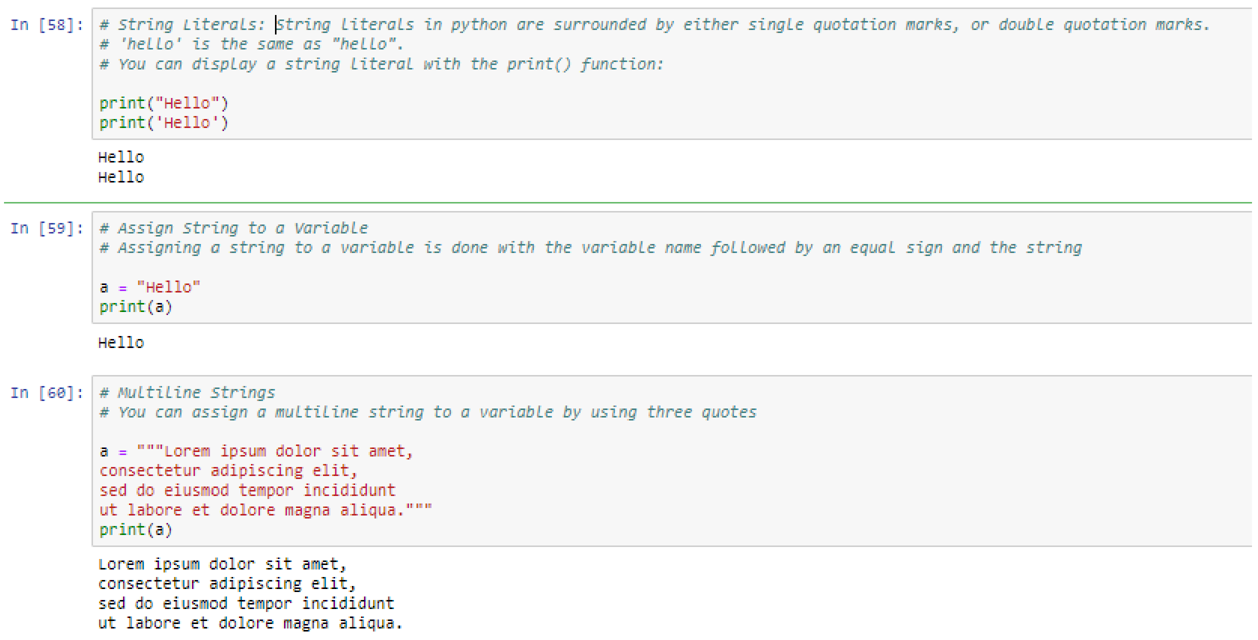


```
In [6]: # Python Strings, which can be used with single, double or triple quotes

In [7]: var1 = 'Hello World!'
        var2 = "Python Programming"

In [8]: var1
Out[8]: 'Hello World!'

In [9]: var2
Out[9]: 'Python Programming'

In [11]: var3 = """this is a long string that is made up of
         several lines and non-printable characters such as
         TAB ( \t ) and they will show up that way when displayed.
         NEWLINEs within the string, whether explicitly given like
         this within the brackets [ \n ], or just a NEWLINE within
         the variable assignment will also show up.
         """

In [12]: var3
Out[12]: 'this is a long string that is made up of\nseveral lines and non-printable characters such as\nTAB ( \t ) and they will show up
         that way when displayed.\nNEWLINEs within the string, whether explicitly given like\nthis within the brackets [ \n ], or just a
         NEWLINE within\nthe variable assignment will also show up.\n'
```

There are different types of string operations in python:

1. String Literals
2. Assign String to a Variable
3. Multiline Strings
4. Strings are Arrays
5. Slicing
6. Negative Indexing
7. String Length
8. String Methods
9. Check String
10. String Concatenation
11. String Format
12. Escape Character
13. String Methods

Find the below images with examples for string operations:



```
In [58]: # String Literals: String literals in python are surrounded by either single quotation marks, or double quotation marks.
         # 'hello' is the same as "hello".
         # You can display a string literal with the print() function:

         print("Hello")
         print('Hello')

         Hello
         Hello
```

```
In [59]: # Assign String to a Variable
         # Assigning a string to a variable is done with the variable name followed by an equal sign and the string

         a = "Hello"
         print(a)

         Hello
```

```
In [60]: # Multiline Strings
         # You can assign a multiline string to a variable by using three quotes

         a = """Lorem ipsum dolor sit amet,
         consectetur adipiscing elit,
         sed do eiusmod tempor incididunt
         ut labore et dolore magna aliqua."""
         print(a)

         Lorem ipsum dolor sit amet,
         consectetur adipiscing elit,
         sed do eiusmod tempor incididunt
         ut labore et dolore magna aliqua.
```

In [61]:
```python
# three single quotes:

a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

In [62]:
```python
# Strings are Arrays:Like many other popular programming languages, strings in Python are arrays of bytes representing unicode
# characters. However, Python does not have a character data type, a single character is simply a string with a length of 1.
# Square brackets can be used to access elements of the string.
# Example : Get the character at position 1 (remember that the first character has the position 0):

a = "Hello, World!"
print(a[1])
```

```
e
```

In [63]:
```python
# Slicing: You can return a range of characters by using the slice syntax.
# Specify the start index and the end index, separated by a colon, to return a part of the string.
# Example: Get the characters from position 2 to position 5 (not included):

b = "Hello, World!"
print(b[2:5])
```

```
llo
```

In [64]:
```python
# Negative Indexing: Use negative indexes to start the slice from the end of the string:
# Example: Get the characters from position 5 to position 1, starting the count from the end of the string

b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

In [65]:
```python
# String Length : To get the length of a string, use the len() function.
# Example: The len() function returns the length of a string

a = "Hello, World!"
print(len(a))
```

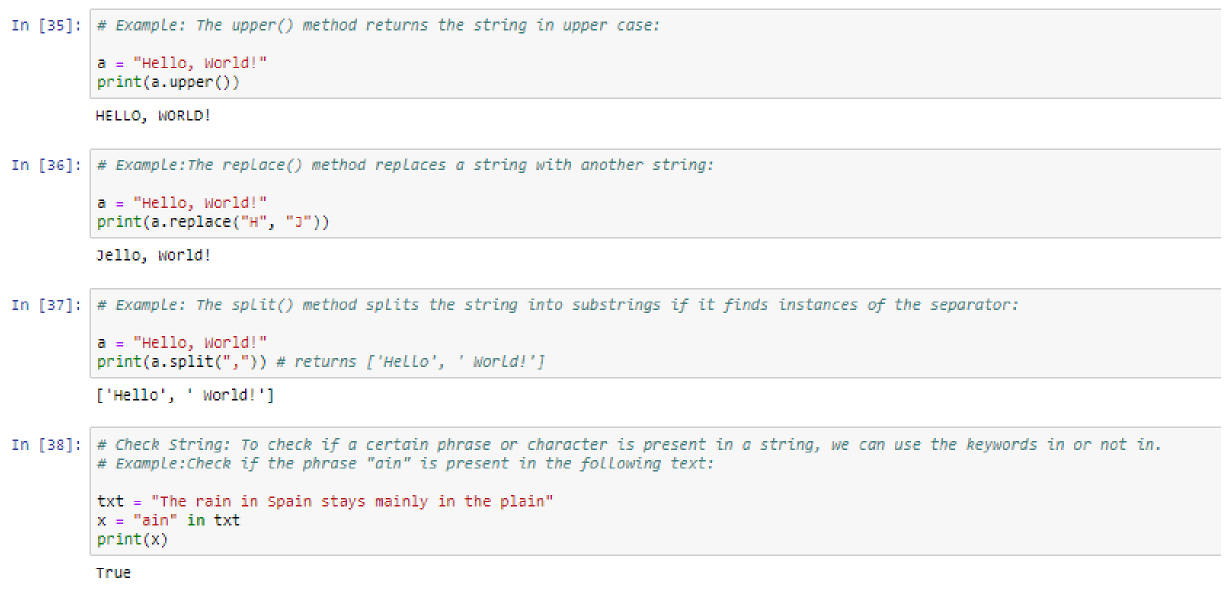
```
13
```

In [66]:
```python
# String Methods: Python has a set of built-in methods that you can use on strings.
# Example: The strip() method removes any whitespace from the beginning or the end

a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

```
Hello, World!
```

In [34]:
```python
# String Methods:The lower() method returns the string in lower case:

a = "Hello, World!"
print(a.lower())
```

```
hello, world!
```

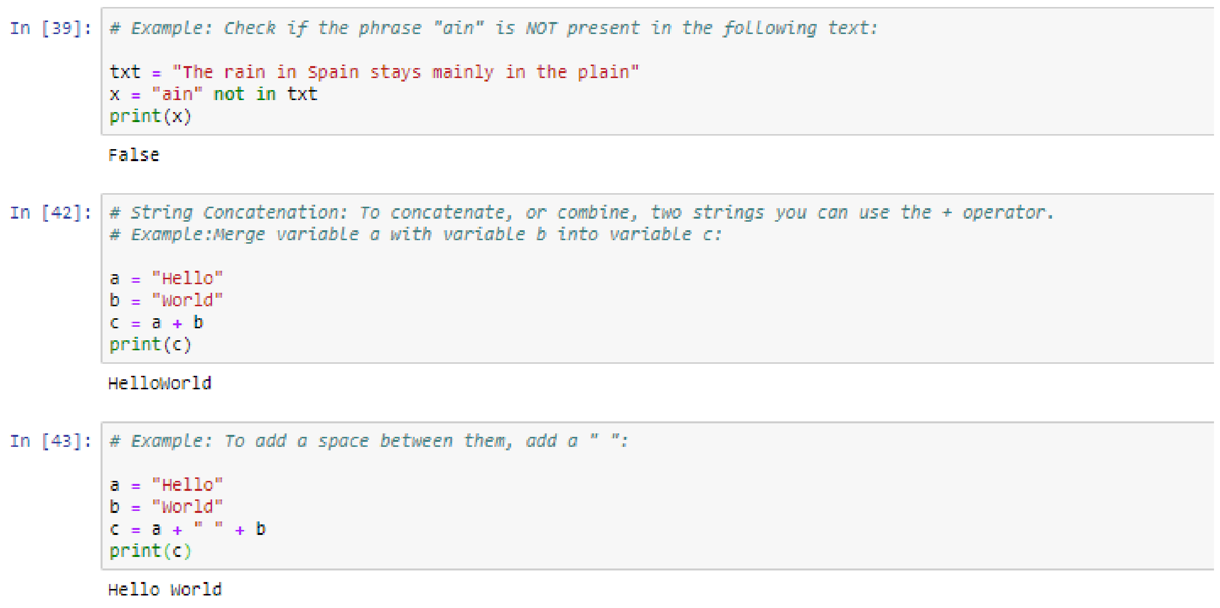


In [35]:
```
# Example: The upper() method returns the string in upper case:

a = "Hello, World!"
print(a.upper())
```
```
HELLO, WORLD!
```

In [36]:
```
# Example:The replace() method replaces a string with another string:

a = "Hello, World!"
print(a.replace("H", "J"))
```
```
Jello, World!
```

In [37]:
```
# Example: The split() method splits the string into substrings if it finds instances of the separator:

a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```
```
['Hello', ' World!']
```

In [38]:
```
# Check String: To check if a certain phrase or character is present in a string, we can use the keywords in or not in.
# Example:Check if the phrase "ain" is present in the following text:

txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x)
```
```
True
```



In [39]:
```
# Example: Check if the phrase "ain" is NOT present in the following text:

txt = "The rain in Spain stays mainly in the plain"
x = "ain" not in txt
print(x)
```
```
False
```

In [42]:
```
# String Concatenation: To concatenate, or combine, two strings you can use the + operator.
# Example:Merge variable a with variable b into variable c:

a = "Hello"
b = "World"
c = a + b
print(c)
```
```
HelloWorld
```

In [43]:
```
# Example: To add a space between them, add a " ":

a = "Hello"
b = "World"
c = a + " " + b
print(c)
```
```
Hello World
```

In [44]:
```
# String Format: As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:
# Example
age = 36
txt = "My name is John, I am " + age
print(txt)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-78c96765afe1> in <module>
      3 # Example
      4 age = 36
----> 5 txt = "My name is John, I am " + age
      6 print(txt)

TypeError: can only concatenate str (not "int") to str
```

In [45]:
```
# But we can combine strings and numbers by using the format() method!
# The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:
# Example: Use the format() method to insert numbers into strings:

age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

```
My name is John, and I am 36
```

In [46]:
```
# The format() method takes unlimited number of arguments, and are placed into the respective placeholders:
# Example

quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

In [47]:
```
# You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:
# Example

quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

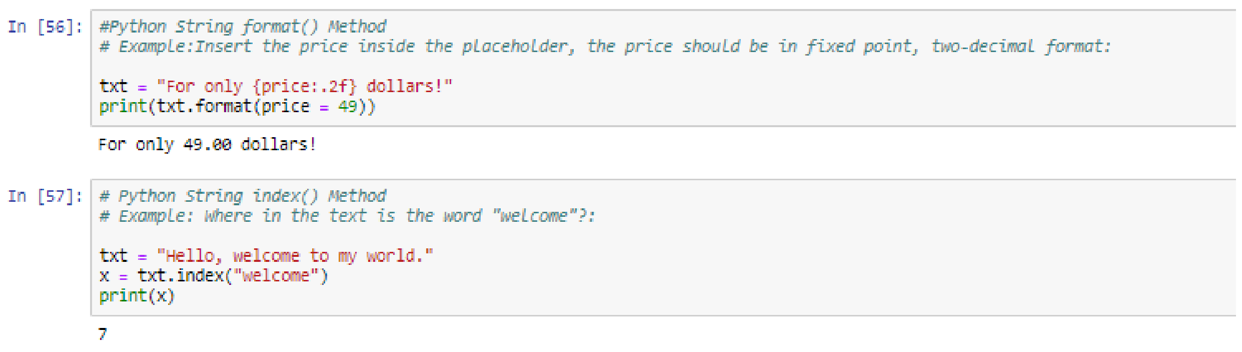
```
I want to pay 49.95 dollars for 3 pieces of item 567.
```

In [49]:
```
# Escape Character : To insert characters that are illegal in a string, use an escape character.
# An escape character is a backslash \ followed by the character you want to insert.
# An example of an illegal character is a double quote inside a string that is surrounded by double quotes:
# Example: You will get an error if you use double quotes inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."
```

```
  File "<ipython-input-49-b8e021153803>", line 9
    txt = "We are the so-called "Vikings" from the north."
                                       ^
SyntaxError: invalid syntax
```

In [53]:
```
# To fix this problem, use the escape character \":
# Example: The escape character allows you to use double quotes when you normally would not be allowed:

txt = "We are the so-called \"Vikings\" from the north."
txt
```

Out[53]:
```
'We are the so-called "Vikings" from the north.'
```



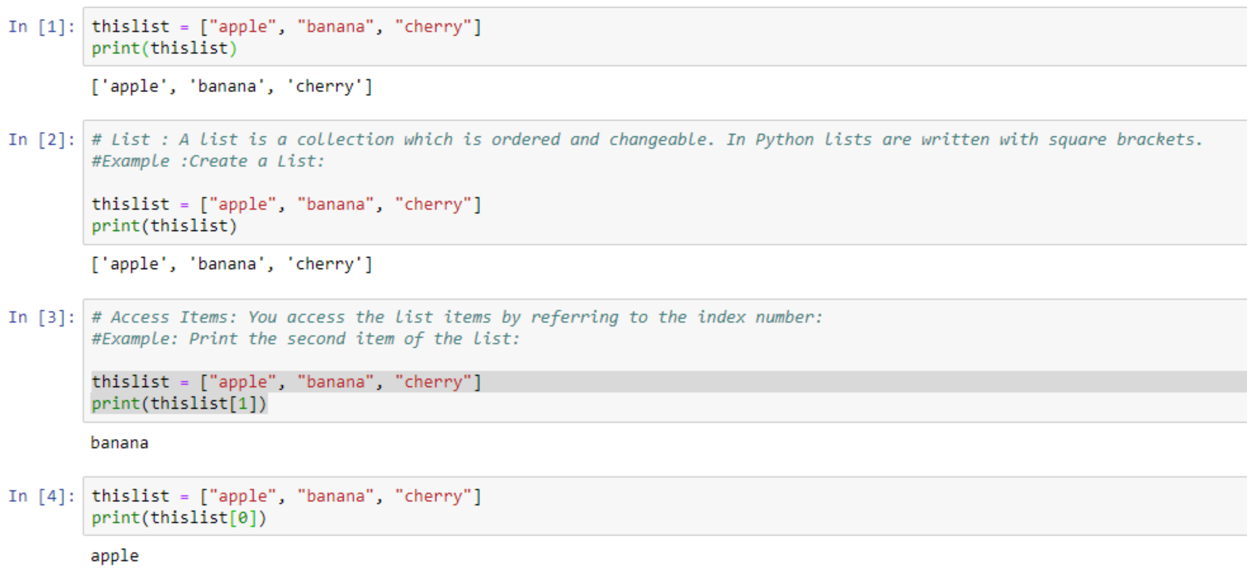
```
In [56]: #Python String format() Method
         # Example:Insert the price inside the placeholder, the price should be in fixed point, two-decimal format:

         txt = "For only {price:.2f} dollars!"
         print(txt.format(price = 49))

         For only 49.00 dollars!

In [57]: # Python String index() Method
         # Example: Where in the text is the word "welcome"?:

         txt = "Hello, welcome to my world."
         x = txt.index("welcome")
         print(x)

         7
```

Data structures in Python

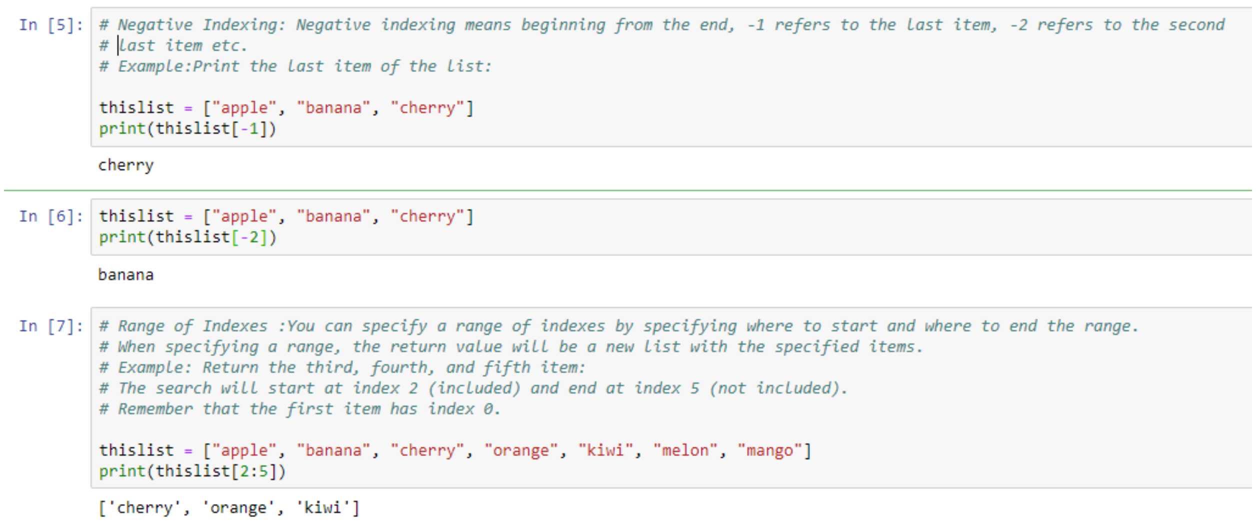
Python Collections (Arrays): There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

List is a collection which is ordered and changeable. Allows duplicate members.



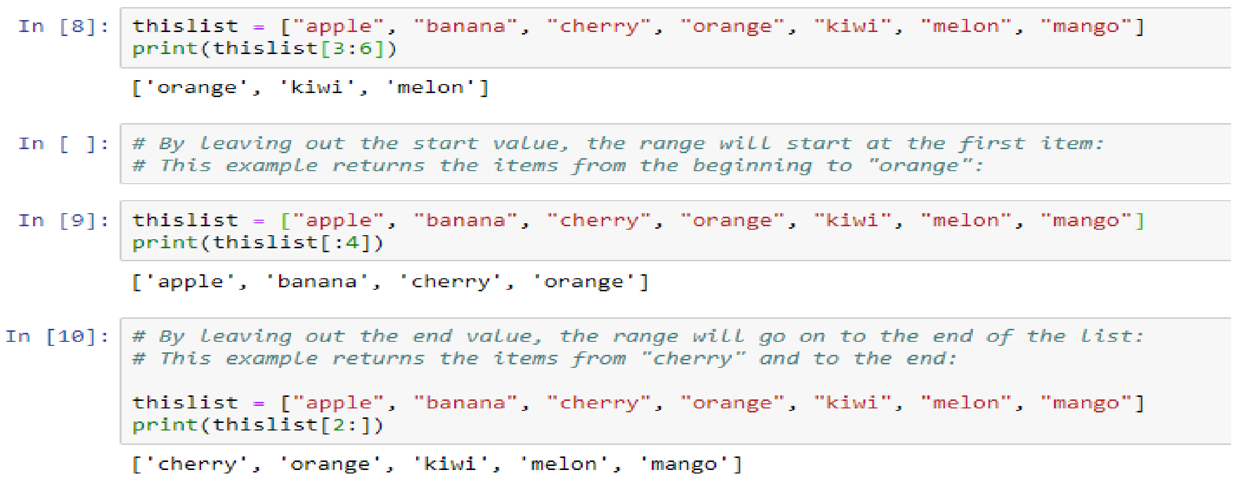
```
In [1]: thislist = ["apple", "banana", "cherry"]
        print(thislist)

        ['apple', 'banana', 'cherry']

In [2]: # List : A list is a collection which is ordered and changeable. In Python lists are written with square brackets.
        #Example :Create a List:

        thislist = ["apple", "banana", "cherry"]
        print(thislist)

        ['apple', 'banana', 'cherry']

In [3]: # Access Items: You access the list items by referring to the index number:
        #Example: Print the second item of the list:

        thislist = ["apple", "banana", "cherry"]
        print(thislist[1])

        banana

In [4]: thislist = ["apple", "banana", "cherry"]
        print(thislist[0])

        apple
```



```
In [5]: # Negative Indexing: Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second
        # last item etc.
        # Example:Print the last item of the list:

        thislist = ["apple", "banana", "cherry"]
        print(thislist[-1])

        cherry
```

```
In [6]: thislist = ["apple", "banana", "cherry"]
        print(thislist[-2])

        banana
```

```
In [7]: # Range of Indexes :You can specify a range of indexes by specifying where to start and where to end the range.
        # When specifying a range, the return value will be a new list with the specified items.
        # Example: Return the third, fourth, and fifth item:
        # The search will start at index 2 (included) and end at index 5 (not included).
        # Remember that the first item has index 0.

        thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
        print(thislist[2:5])

        ['cherry', 'orange', 'kiwi']
```

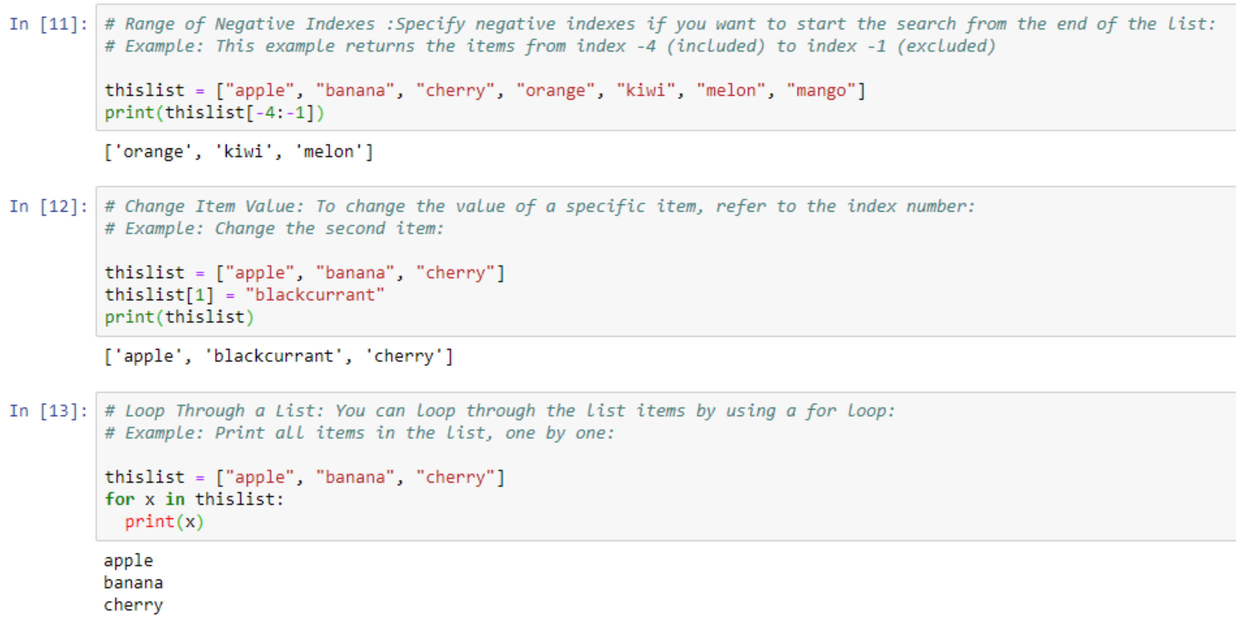


```
In [8]: thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
        print(thislist[3:6])

        ['orange', 'kiwi', 'melon']
```

```
In [ ]: # By leaving out the start value, the range will start at the first item:
        # This example returns the items from the beginning to "orange":
```

```
In [9]: thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
        print(thislist[:4])

        ['apple', 'banana', 'cherry', 'orange']
```

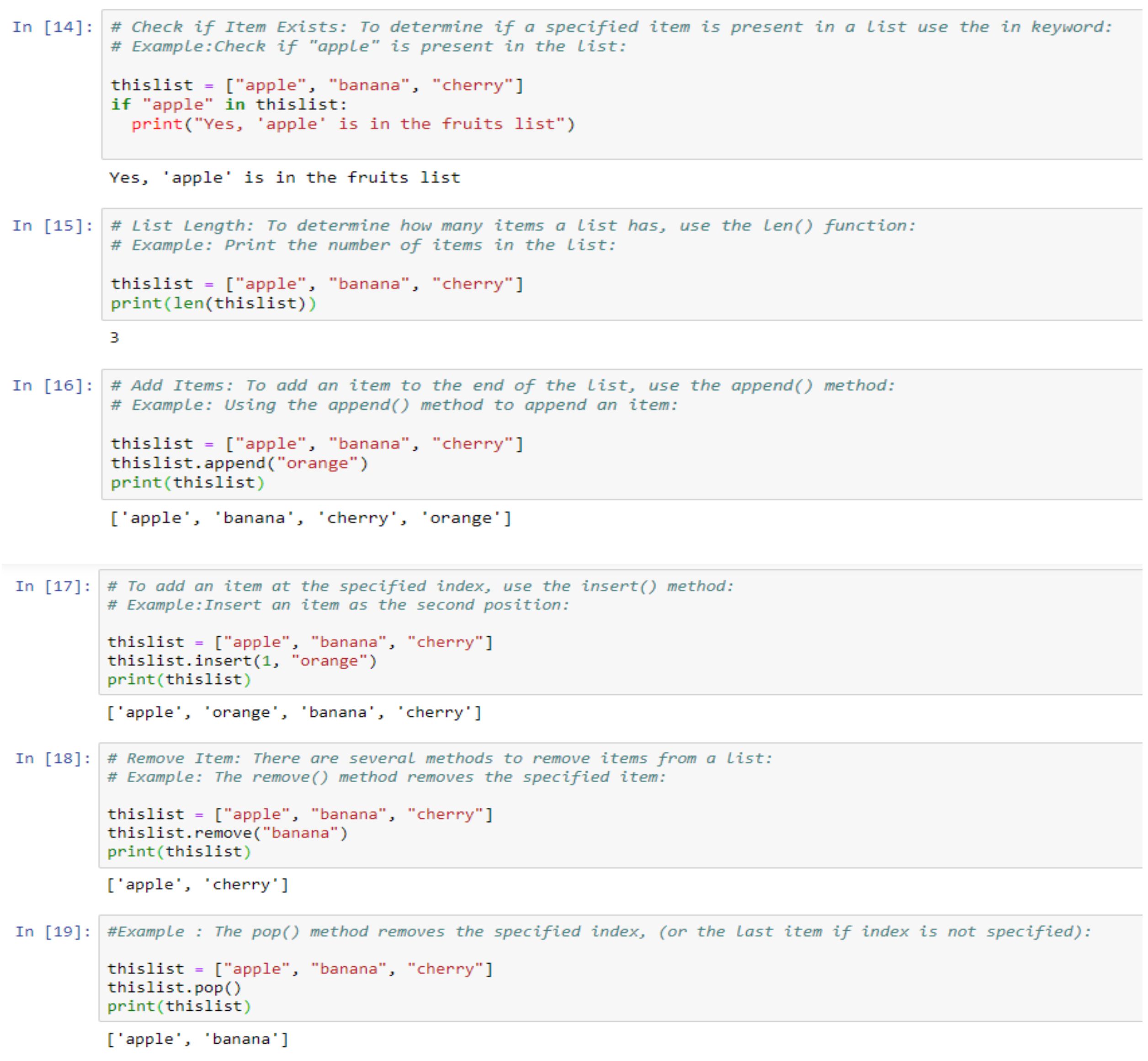
```
In [10]: # By leaving out the end value, the range will go on to the end of the list:
         # This example returns the items from "cherry" and to the end:

         thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
         print(thislist[2:])

         ['cherry', 'orange', 'kiwi', 'melon', 'mango']
```



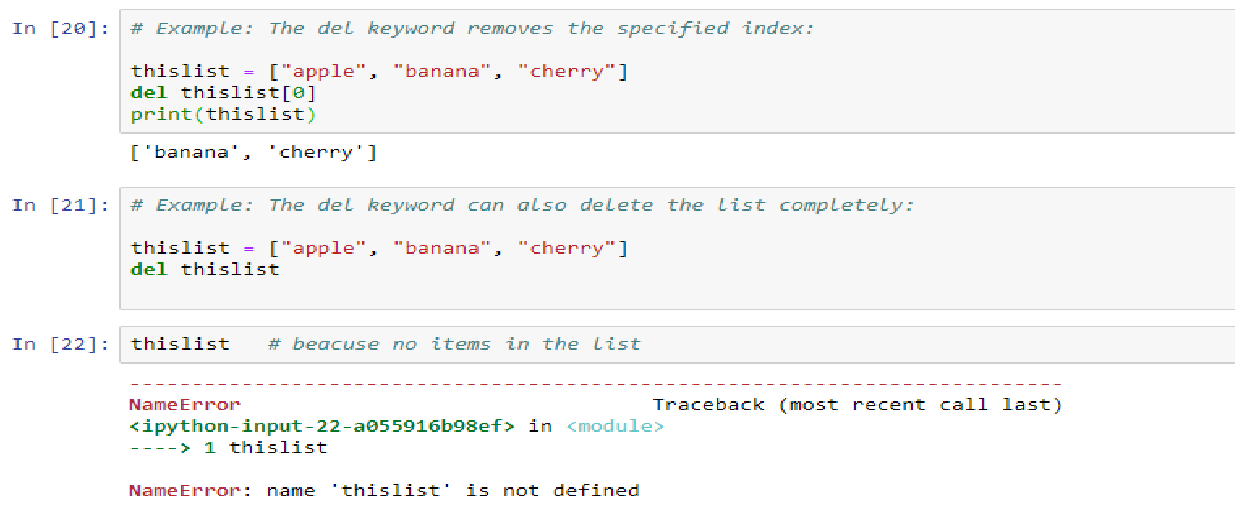
```
In [11]: # Range of Negative Indexes :Specify negative indexes if you want to start the search from the end of the list:
         # Example: This example returns the items from index -4 (included) to index -1 (excluded)

         thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
         print(thislist[-4:-1])

         ['orange', 'kiwi', 'melon']
```

```
In [12]: # Change Item Value: To change the value of a specific item, refer to the index number:
         # Example: Change the second item:

         thislist = ["apple", "banana", "cherry"]
         thislist[1] = "blackcurrant"
         print(thislist)

         ['apple', 'blackcurrant', 'cherry']
```

```
In [13]: # Loop Through a List: You can loop through the list items by using a for loop:
         # Example: Print all items in the list, one by one:

         thislist = ["apple", "banana", "cherry"]
         for x in thislist:
           print(x)

         apple
         banana
         cherry
```



```
In [14]: # Check if Item Exists: To determine if a specified item is present in a list use the in keyword:
         # Example:Check if "apple" is present in the list:

         thislist = ["apple", "banana", "cherry"]
         if "apple" in thislist:
           print("Yes, 'apple' is in the fruits list")

         Yes, 'apple' is in the fruits list

In [15]: # List Length: To determine how many items a list has, use the len() function:
         # Example: Print the number of items in the list:

         thislist = ["apple", "banana", "cherry"]
         print(len(thislist))

         3

In [16]: # Add Items: To add an item to the end of the list, use the append() method:
         # Example: Using the append() method to append an item:

         thislist = ["apple", "banana", "cherry"]
         thislist.append("orange")
         print(thislist)

         ['apple', 'banana', 'cherry', 'orange']

In [17]: # To add an item at the specified index, use the insert() method:
         # Example:Insert an item as the second position:

         thislist = ["apple", "banana", "cherry"]
         thislist.insert(1, "orange")
         print(thislist)

         ['apple', 'orange', 'banana', 'cherry']

In [18]: # Remove Item: There are several methods to remove items from a list:
         # Example: The remove() method removes the specified item:

         thislist = ["apple", "banana", "cherry"]
         thislist.remove("banana")
         print(thislist)

         ['apple', 'cherry']

In [19]: #Example : The pop() method removes the specified index, (or the last item if index is not specified):

         thislist = ["apple", "banana", "cherry"]
         thislist.pop()
         print(thislist)

         ['apple', 'banana']
```



```
In [20]: # Example: The del keyword removes the specified index:

         thislist = ["apple", "banana", "cherry"]
         del thislist[0]
         print(thislist)

         ['banana', 'cherry']

In [21]: # Example: The del keyword can also delete the list completely:

         thislist = ["apple", "banana", "cherry"]
         del thislist

In [22]: thislist   # beacuse no items in the list

         ---------------------------------------------------------------------------
         NameError                                 Traceback (most recent call last)
         <ipython-input-22-a055916b98ef> in <module>
         ----> 1 thislist

         NameError: name 'thislist' is not defined
```

In [23]:
```python
# Example: The clear() method empties the list:

thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

In [24]:
```python
thislist
```

Out[24]:
```
[]
```

In [25]:
```python
# Copy a List: You cannot copy a list simply by typing list2 = list1, because: list2 will only be a
# reference to list1, and changes made in list1 will automatically also be made in list2.
# There are ways to make a copy, one way is to use the built-in List method copy().
# Example: Make a copy of a list with the copy() method:

thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

In [26]:
```python
mylist
```

Out[26]:
```
['apple', 'banana', 'cherry']
```

In [26]:
```python
mylist
```

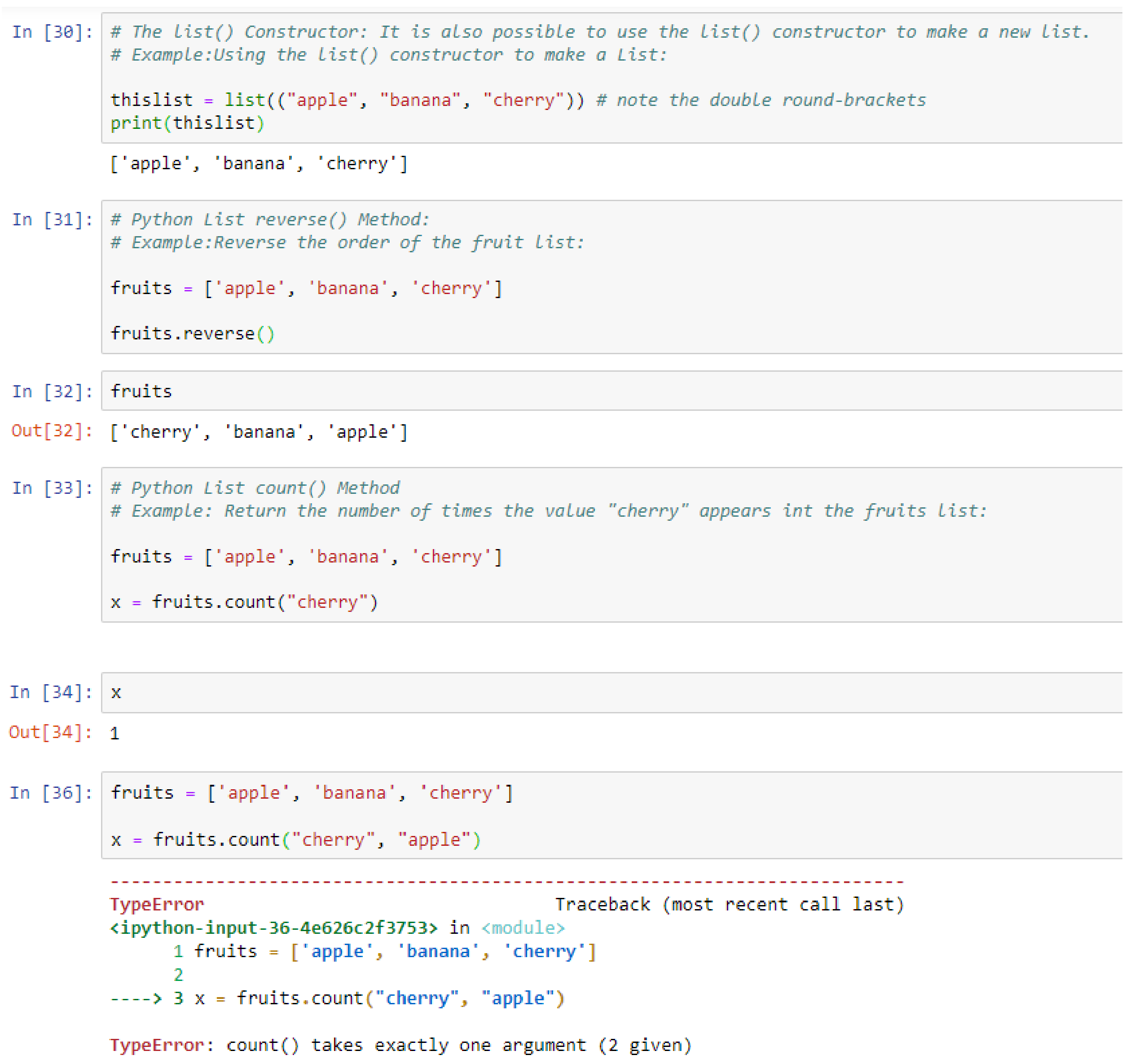
Out[26]:
```
['apple', 'banana', 'cherry']
```

In [27]:
```python
# Join Two Lists: There are several ways to join, or concatenate, two or more lists in Python.
# One of the easiest ways are by using the + operator.
# Example: Join two list:

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

In [29]:
```python
# Another way to join two lists are by appending all the items from list2 into list1, one by one:
# Example: Append list2 into list1:

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```



```
In [30]: # The list() Constructor: It is also possible to use the list() constructor to make a new list.
         # Example:Using the list() constructor to make a List:

         thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
         print(thislist)

         ['apple', 'banana', 'cherry']

In [31]: # Python List reverse() Method:
         # Example:Reverse the order of the fruit list:

         fruits = ['apple', 'banana', 'cherry']

         fruits.reverse()

In [32]: fruits

Out[32]: ['cherry', 'banana', 'apple']

In [33]: # Python List count() Method
         # Example: Return the number of times the value "cherry" appears int the fruits list:

         fruits = ['apple', 'banana', 'cherry']

         x = fruits.count("cherry")


In [34]: x

Out[34]: 1

In [36]: fruits = ['apple', 'banana', 'cherry']

         x = fruits.count("cherry", "apple")

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-36-4e626c2f3753> in <module>
               1 fruits = ['apple', 'banana', 'cherry']
               2
         ----> 3 x = fruits.count("cherry", "apple")

         TypeError: count() takes exactly one argument (2 given)
```