Python is a high-level, interpreted programming language that is widely used for many different purposes, from web development to scientific computing to machine learning.

There are several reasons why Python is a great choice for many different types of projects:

1. **Easy to learn:** Python has a relatively simple syntax, which makes it easier for beginners to learn compared to other programming languages.
2. **Versatile:** Python can be used for a wide range of applications, including web development, data analysis, artificial intelligence, and more.
3. **Large and active community:** There is a large and active community of Python developers, which means that there are many resources available for learning and solving problems.
4. **Plenty of libraries**: Python has a large number of libraries, including NumPy, pandas, and Matplotlib, which can be used to perform complex tasks with just a few lines of code.
5. **Cross-platform compatibility:** Python can run on multiple operating systems, including Windows, macOS, and Linux, making it a good choice for projects that need to be run on multiple platforms.

**Understanding Variables in Python**

Assignment operator (=) to assign new variables in python whether Boolean, float or integer.

Eg. Middle\_name =”josef Wambua”

Print(middle\_name)

# assigning a new value to middle\_name

Middle\_name = “Muindi”

Print(middle\_name)

Output

Josef Wambua

Muindi

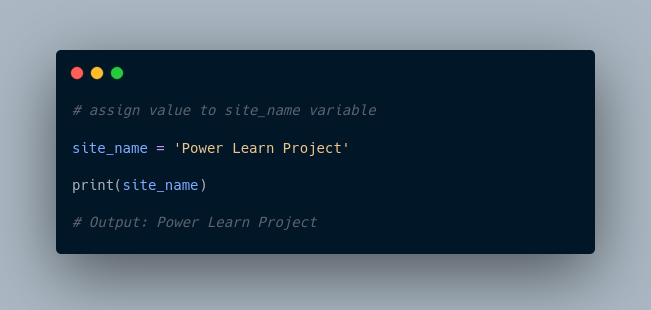
In programming, a variable is a container (storage area) to hold data. For example,

number = 10

Here, number is the variable storing the value **10**.

## Assigning values to Variables in Python

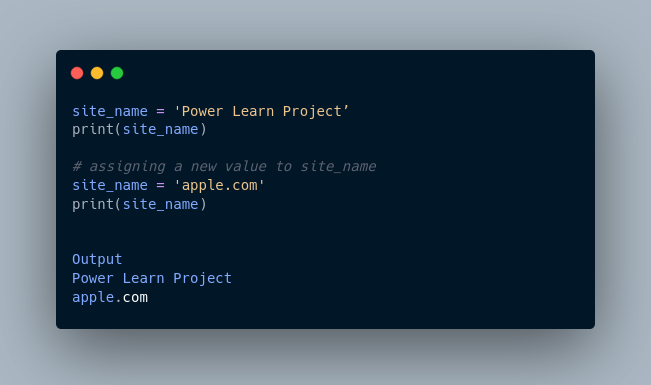
As we can see from the above example, we use the assignment operator = to assign a value to a variable.

﻿

In the above example, we assigned the value ‘Power Learn Project’ to the site\_name variable. Then, we printed out the value assigned to site\_name.

**Note**: Python is a type-inferred language, so you don't have to explicitly define the variable type. It automatically knows that Power Learn Projects is a string and declares the site\_name variable as a string.

## Changing the Value of a Variable in Python

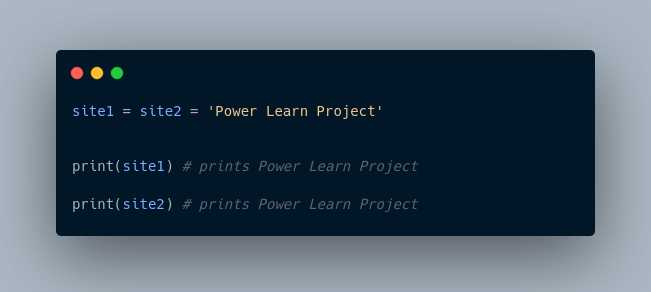


Here, the value of site\_name is changed from ‘Power Learn Project’ to 'apple.com'.

## Example: Assigning multiple values to multiple variables



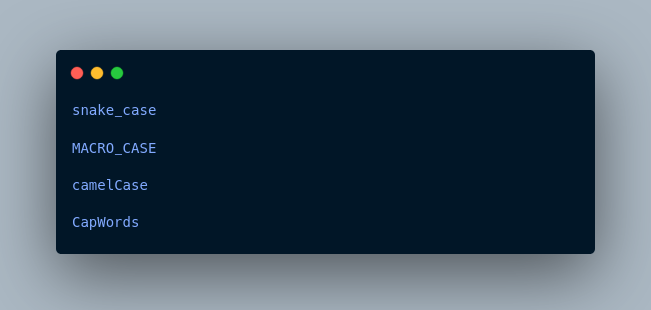
If we want to assign the same value to multiple variables at once, we can do this as:



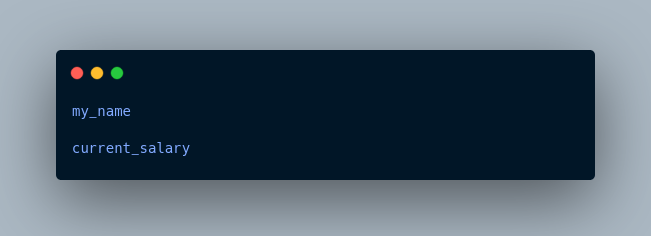
Here, we have assigned the same string value 'Power Learn Project' to both the variables site1 and site2.

## Rules for Naming Python Variables

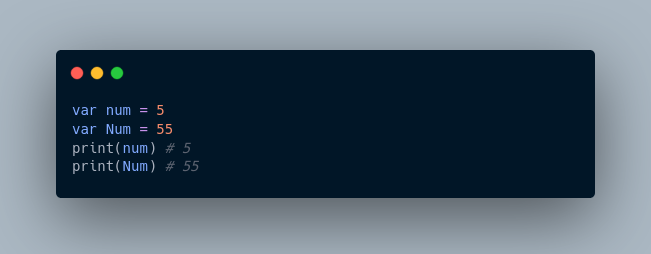
* Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (**A to Z**) or digits (**0 to 9**) or an underscore (**\_**). For example:



* Create a name that makes sense. For example, vowel makes more sense than v.
* If you want to create a variable name having two words, use underscore to separate them. For example:



* Python is case-sensitive. So num and Num are different variables. For example,



* Avoid using keywords like if, True, class, etc. as variable names.

More Resources:

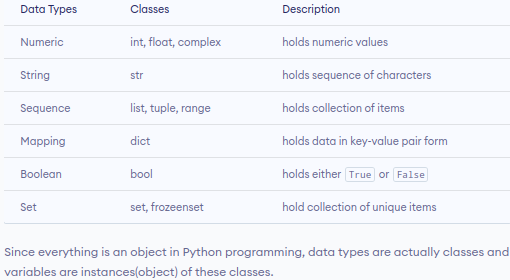
1. <https://realpython.com/python-variables/>
2. <https://www.simplilearn.com/tutorials/python-tutorial/python-variables>
3. <https://www.guru99.com/variables-in-python.html>
4. <https://www.tutorialspoint.com/python/python_variables.htm>

In computer programming, data types specify the type of data that can be stored inside a variable. For example,

num = 24

Here, **24** (an integer) is assigned to the num variable. So the data type of num is of the int class.

**Python Data Types**



Since everything is an object in Python programming, data types are actually classes and variables are instances(object) of these classes.

**Python Numeric Data type**

In Python, the numeric data type is used to hold numeric values.

Integers, floating-point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python.

* int - holds signed integers of non-limited length.
* float - holds floating decimal points and it's accurate up to **15** decimal places.
* complex - holds complex numbers.

We can use the type() function to know which class a variable or a value belongs to.

Let's see an example,



In the above example, we have created three variables named num1, num2 and num3 with values **5**, **5.0**, and 1+2j respectively.

We have also used the type() function to know which class a certain variable belongs to.

Since,

* **5** is an integer value, type() returns int as the class of num1 i.e <class 'int'>
* **2.0** is a floating value, type() returns float as the class of num2 i.e <class 'float'>
* 1 + 2j is a complex number, type() returns complex as the class of num3 i.e <class 'complex'>

**Python List Data Type**

A list is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ]. For example,



Here, we have created a list named languages with **3** string values inside it.

**Access List Items**

To access items from a list, we use the index number **(0, 1, 2 ...)**. For example,



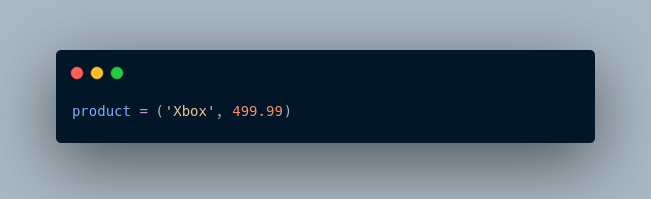
In the above example, we have used the index values to access items from the languages list.

* languages[0] - access first item from languages i.e. Swift
* languages[2] - access third item from languages i.e. Python

**Python Tuple Data Type**

A tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

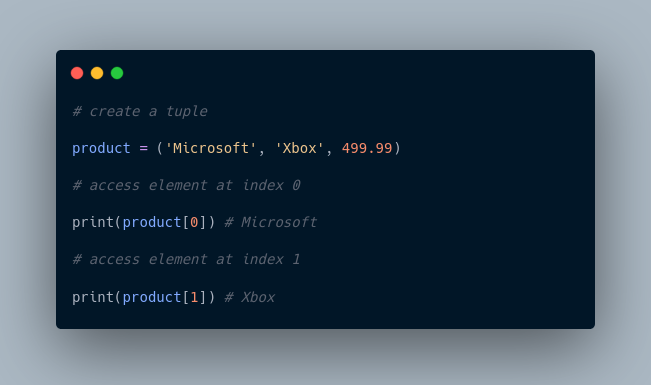
In Python, we use the parentheses () to store items of a tuple. For example,



Here, product is a tuple with a string value Xbox and integer value **499.99**.

**Access Tuple Items**

Similar to lists, we use the index number to access tuple items in Python. For example,



**Python String Data Type**

String is a sequence of characters represented by either single or double quotes. For example,



In the above example, we have created string-type variables: name and message with values 'Python' and 'Python for beginners' respectively.

**Python Set Data Type**

The Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces { }. For example,



Here, we have created a set named student\_info with **5** integer values.

Since sets are unordered collections, indexing has no meaning. Hence, the slicing operator [] does not work.

**Python Dictionary Data Type**

Python dictionary is an ordered collection of items. It stores elements in key/value pairs.

Here, keys are unique identifiers that are associated with each value.

Let's see an example,



In the above example, we have created a dictionary named capital\_city. Here,

1. **Keys** are 'Nepal', 'Italy', 'England'
2. **Values** are 'Kathmandu', 'Rome', 'London'

**Access Dictionary Values Using Keys**

We use keys to retrieve the respective value. But not the other way around. For example,



Here, we have accessed values using keys from the capital\_city dictionary.

Since 'Nepal' is key, capital\_city['Nepal'] accesses its respective value i.e. Kathmandu

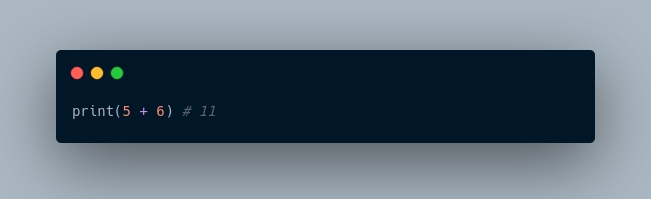
However, 'Kathmandu' is the value for the 'Nepal' key, so capital\_city['Kathmandu'] throws an error message.

More Resources:

1. <https://www.geeksforgeeks.org/python-data-types/>
2. <https://realpython.com/python-data-types/>
3. <https://www.digitalocean.com/community/tutorials/python-data-types>
4. <https://jakevdp.github.io/PythonDataScienceHandbook/02.01-understanding-data-types.html>

In conclusion:  
Sets are enclosed using {}, tuples using (), list [],.. dictionary using parenthesis too

Operators are special symbols that perform operations on variables and values. For example,



Here, + is an operator that adds two numbers: **5** and **6**.

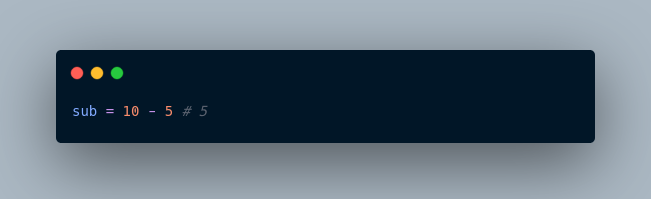
## ****Types of Python Operators****

Here's a list of different types of Python operators that we will learn in this tutorial.

1. Arithmetic operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. Special Operators

### 1. **Python Arithmetic Operators**

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc. For example,



Here, - is an arithmetic operator that subtracts two values or variables.

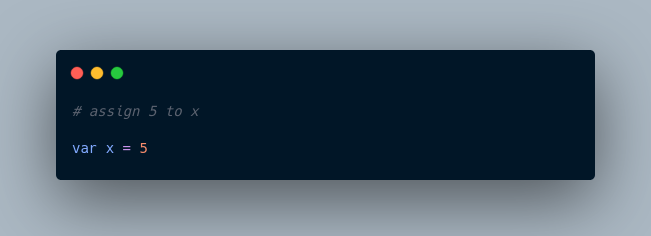
****

In the above example, we have used multiple arithmetic operators,

* + to add a and b
* - to subtract b from a
* \* to multiply a and b
* / to divide a by b
* // to floor divide a by b
* % to get the remainder
* \*\* to get a to the power b

### **2. Python Assignment Operators**

Assignment operators are used to assign values to variables. For example,



Here, = is an assignment operator that assigns 5 to x.

Here's a list of different assignment operators available in Python.

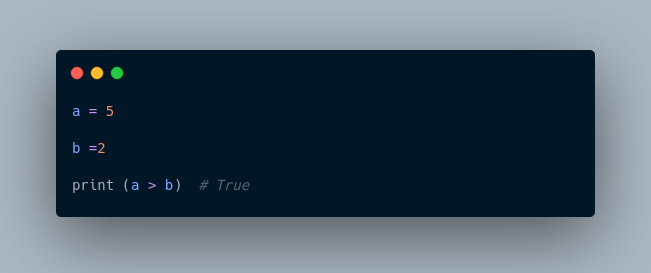
****

Here, we have used the += operator to assign the sum of a and b to a.

Similarly, we can use any other assignment operators according to the need.

### **3. Python Comparison Operators**

Comparison operators compare two values/variables and return a boolean result: True or False. For example,



Here, the > comparison operator is used to compare whether a is greater than b or not.

****

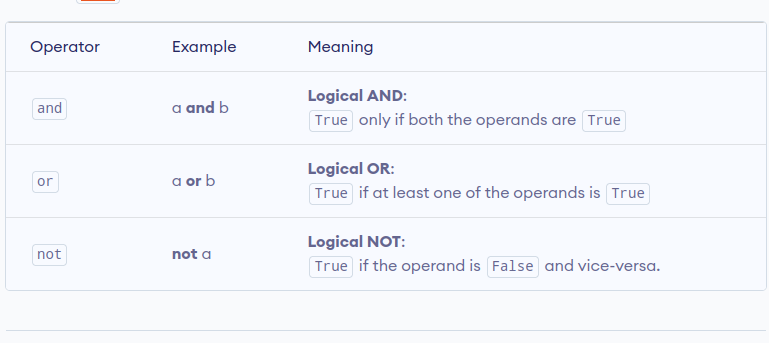
**Note:** Comparison operators are used in decision-making and loops. We'll discuss more of the comparison operator and decision-making in later tutorials.

### **4. Python Logical Operators**

Logical operators are used to check whether an expression is True or False. They are used in decision-making. For example,



Here, and is the logical operator **AND**. Since both a > 2 and b >= 6 are True, the result is True.

****

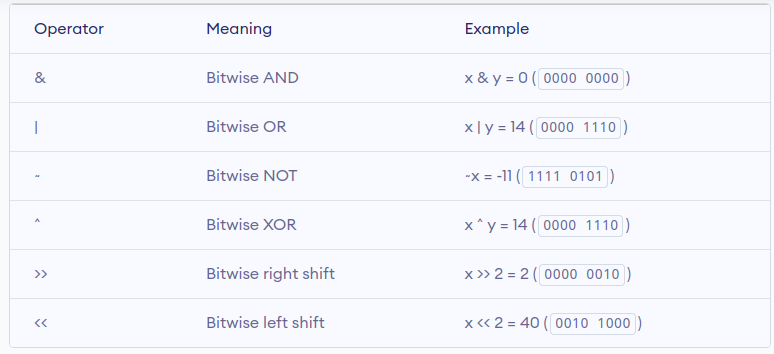
**Note**: Here is the truth table for these logical operators.

### **5. Python Bitwise operators**

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example, **2** is 10 in binary and **7** is 111.

**In the table below:** Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

****

## ****6. Python Special operators****

Python language offers some special types of operators like the **identity** operator and the **membership** operator. They are described below with examples.

### **Identity operators**

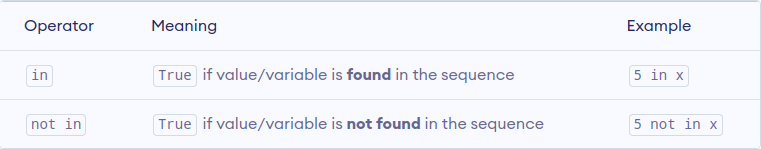
In Python, is and is not being used to check if two values are located on the same part of the memory. Two equal variables do not imply that they are identical.

****

## ****Membership operators****

In Python, in and not in are the membership operators. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set, and dictionary).

In a dictionary, we can only test for the presence of a key, not the value.

****

### **Example 5: Membership operators in Python**

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case-sensitive).

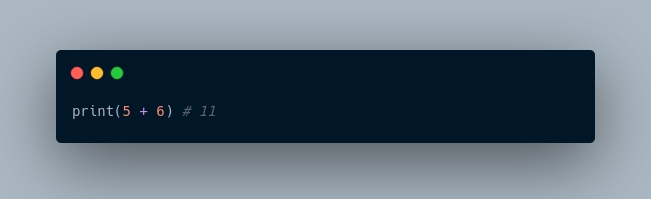
﻿

Similarly, 1 is the key, and 'a' is the value in the dictionary y. Hence, 'a' in y returns False.

More Resources:

1. <https://www.w3schools.com/python/python_operators.asp>
2. <https://www.freecodecamp.org/news/basic-operators-in-python-with-examples/>
3. <https://www.geeksforgeeks.org/python-operators/>

Operators are special symbols that perform operations on variables and values. For example,



Here, + is an operator that adds two numbers: **5** and **6**.

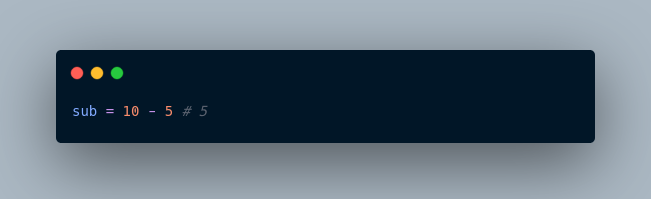
## ****Types of Python Operators****

Here's a list of different types of Python operators that we will learn in this tutorial.

1. Arithmetic operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. Special Operators

### 1. **Python Arithmetic Operators**

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc. For example,



Here, - is an arithmetic operator that subtracts two values or variables.

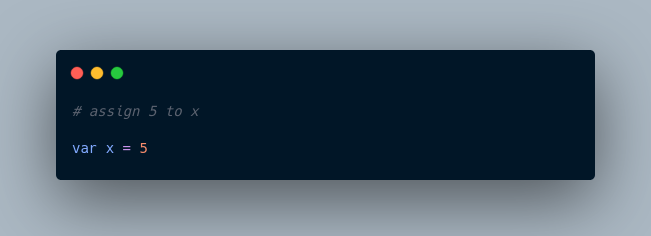
****

In the above example, we have used multiple arithmetic operators,

* + to add a and b
* - to subtract b from a
* \* to multiply a and b
* / to divide a by b
* // to floor divide a by b
* % to get the remainder
* \*\* to get a to the power b

### **2. Python Assignment Operators**

Assignment operators are used to assign values to variables. For example,



Here, = is an assignment operator that assigns 5 to x.

Here's a list of different assignment operators available in Python.

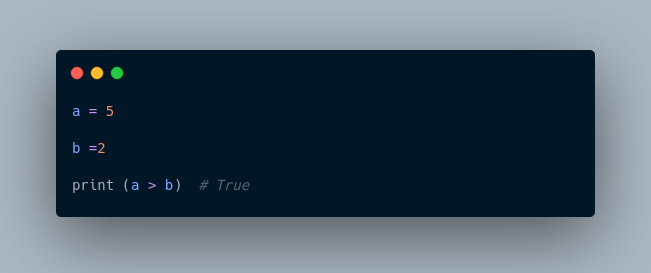
****

Here, we have used the += operator to assign the sum of a and b to a.

Similarly, we can use any other assignment operators according to the need.

### **3. Python Comparison Operators**

Comparison operators compare two values/variables and return a boolean result: True or False. For example,



Here, the > comparison operator is used to compare whether a is greater than b or not.

****

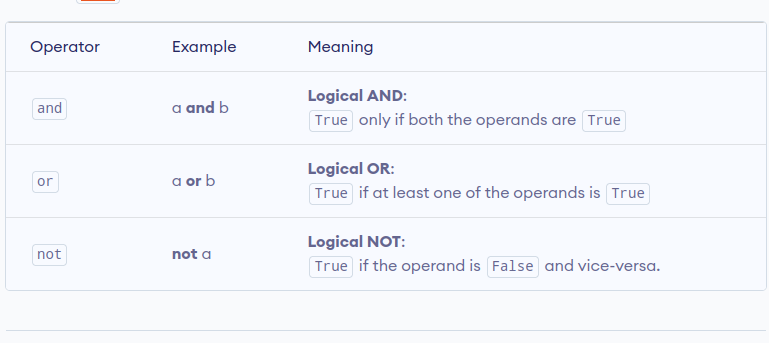
**Note:** Comparison operators are used in decision-making and loops. We'll discuss more of the comparison operator and decision-making in later tutorials.

### **4. Python Logical Operators**

Logical operators are used to check whether an expression is True or False. They are used in decision-making. For example,



Here, and is the logical operator **AND**. Since both a > 2 and b >= 6 are True, the result is True.

****

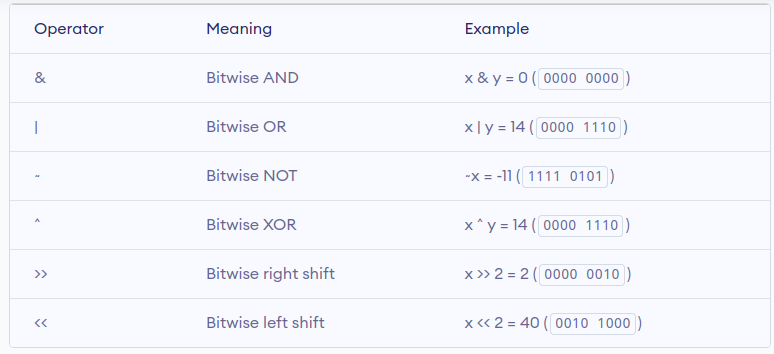
**Note**: Here is the truth table for these logical operators.

### **5. Python Bitwise operators**

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example, **2** is 10 in binary and **7** is 111.

**In the table below:** Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

****

## ****6. Python Special operators****

Python language offers some special types of operators like the **identity** operator and the **membership** operator. They are described below with examples.

### **Identity operators**

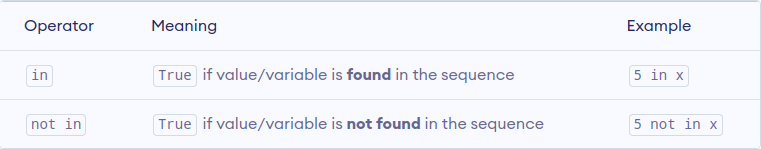
In Python, is and is not being used to check if two values are located on the same part of the memory. Two equal variables do not imply that they are identical.

****

## ****Membership operators****

In Python, in and not in are the membership operators. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set, and dictionary).

In a dictionary, we can only test for the presence of a key, not the value.

****

### **Example 5: Membership operators in Python**

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case-sensitive).

﻿

Similarly, 1 is the key, and 'a' is the value in the dictionary y. Hence, 'a' in y returns False.

More Resources:

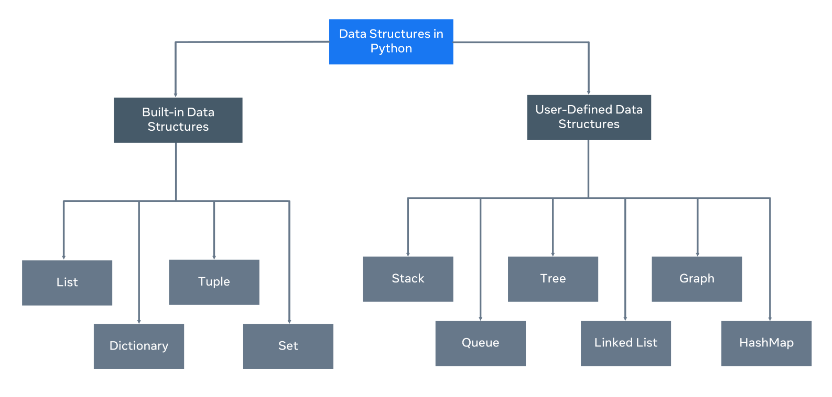
1. <https://www.w3schools.com/python/python_operators.asp>
2. <https://www.freecodecamp.org/news/basic-operators-in-python-with-examples/>
3. <https://www.geeksforgeeks.org/python-operators/>

**WEEK 2**

**Data structures**

So far, you have only stored small bits of data in a variable. This was either an integer, Boolean or a string.

But what happens if you need to work with more complex information, such as a collection of data like a list of people or a list of companies? Data structures are designed for this very purpose.

****

A data structure allows you to organize and arrange your data to perform operations on them. Python has the following built-in data structures: **List**, **dictionary**, **tuple** and **set**. These are all considered **non-primitive** data structures, meaning they are classed as objects, we will explore this more.

Along with the built-in data structures, Python allows users to create their own. Data structures such as Stacks, Queues and Trees can all be created by the user.

Each data structure can be designed to solve a particular problem or optimize a current solution to make it much more performant.

## Mutability and Immutability

Data Structures can be mutable or immutable.

**Mutability** refers to data inside the data structure that can be modified. For example, you can either change, update, or delete the data when needed. A list is an example of a mutable data structure.

An **immutable** data structure will not allow modification once the data has been set. The tuple is an example of an immutable data structure.

**Lists and Tuples**

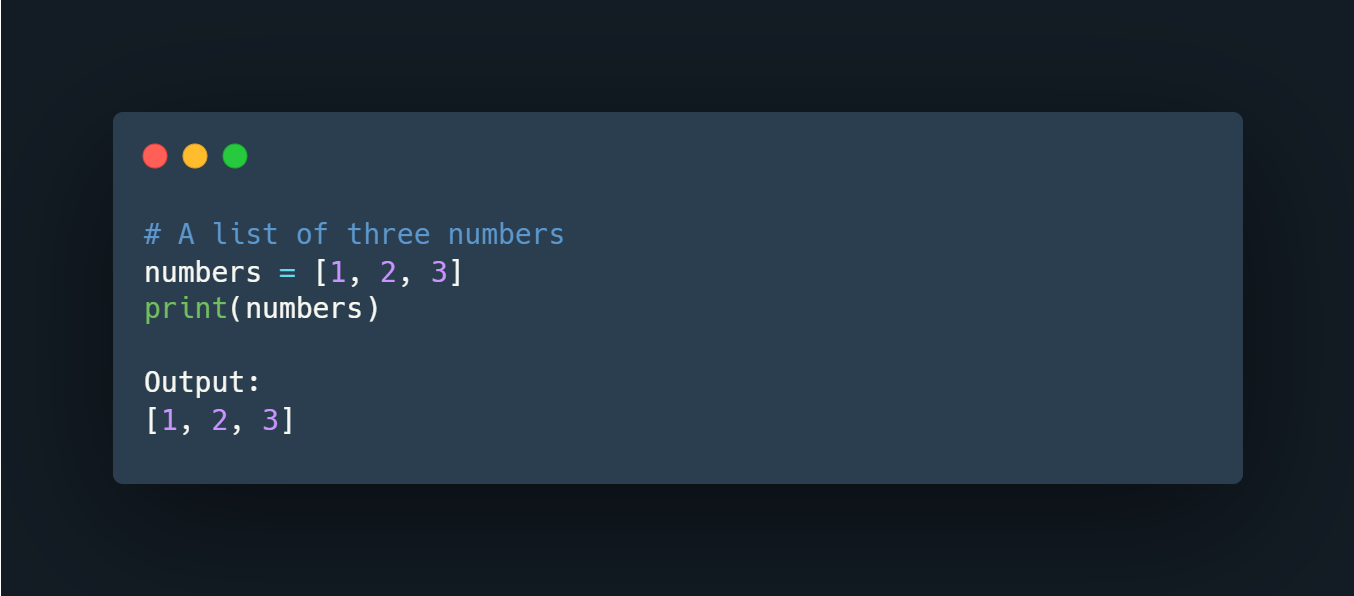
In Python, lists are used to store multiple data at once. For example,

Suppose we need to record the ages of **5** students. Instead of creating **5** separate variables, we can simply create a list:

Lists Elements

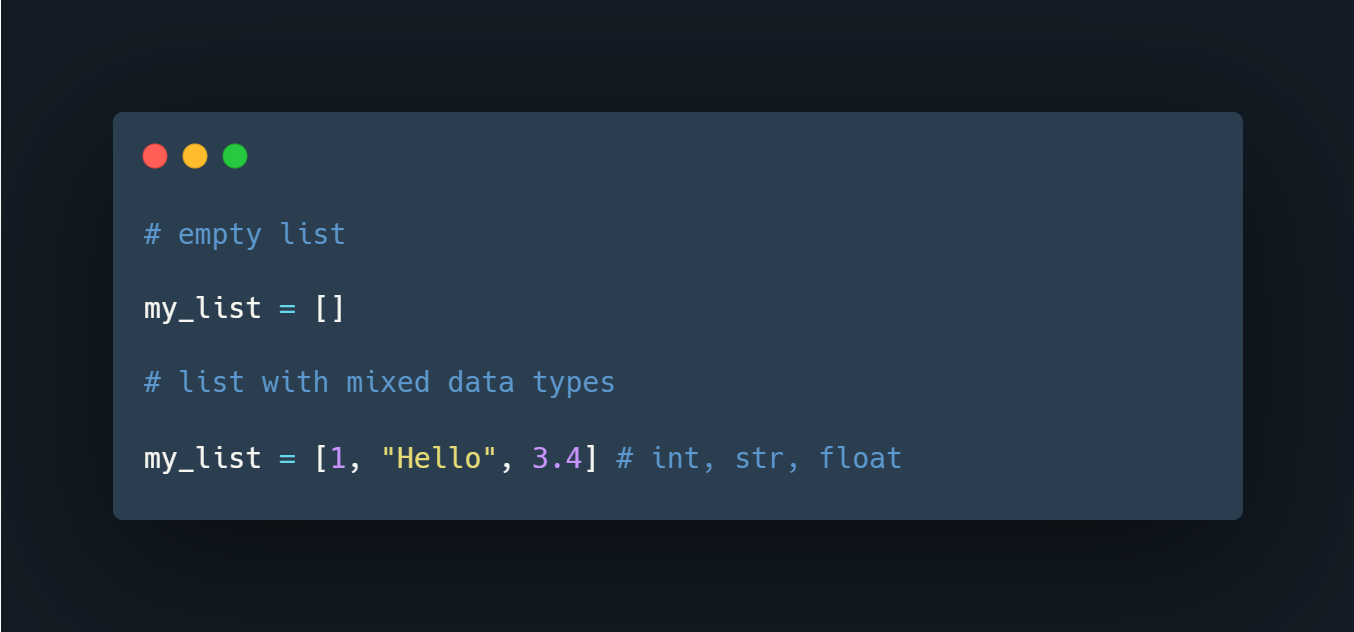
## ****Create a Python List****

A list is created in Python by placing items inside [], separated by commas. For example,



Here, we have created a list named numbers with **3** integer items.

A list can have any number of items and they may be of different types (integer, float, string, etc.). For example,



## ****Access Python List Elements****

In Python, each item in a list is associated with a number. The number is known as a list index.

We can access elements of an array using the index number **(0, 1, 2 …)**. For example,



In the above example, we have created a list named languages.



List Indexing in Python

Here, we can see each list item is associated with the index number. And, we have used the index number to access the items.

**Note:** The list index always starts with **0**. Hence, the first element of a list is present at index **0**, not **1**.

## ****Slicing of a Python List****

In Python, it is possible to access a section of items from the list using the slicing operator :, not just a single item. For example,



Here,

* my\_list[2:5] returns a list with items from index **2** to index **4**.
* my\_list[5:] returns a list with items from index **1** to the end.
* my\_list[:] returns all list items

**Note**: When we slice lists, the start index is inclusive but the end index is exclusive.

## ****Add Elements to a Python List****

Python List provides different methods to add items to a list.

**1. Using append()**

The append() method adds an item at the end of the list.

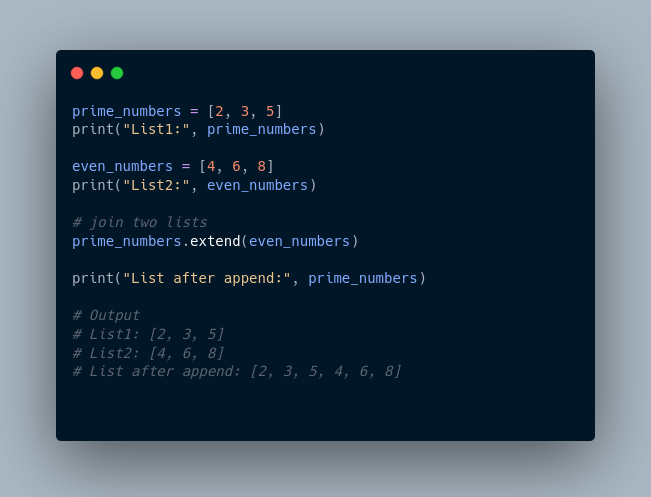
For example,



Here, append() adds **32** at the end of the array.

**2. Using extend()**

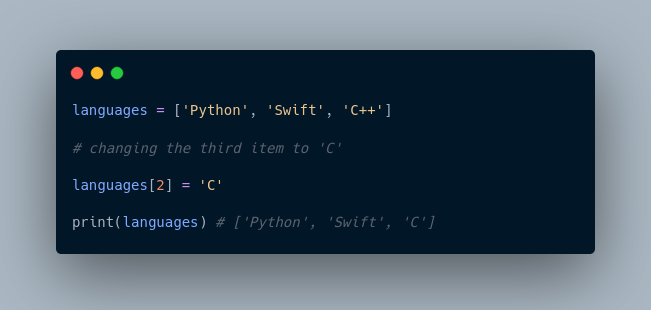
We use the extend() method to add all items of one list to another. For example,



In the above example, we have two lists named prime\_numbers and even\_numbers. Notice the statement,

## ****Change List Items****

Python lists are mutable. Meaning lists are changeable. And, we can change items of a list by assigning new values using = operator. For example,



## ****Remove an Item From a List****

**1. Using del()**

In Python, we can use the del statement to remove one or more items from a list. For example,



**2. Using remove()**

We can also use the remove() method to delete a list item.

For example:



Here, languages.remove('Python') removes 'Python' from the languages list.

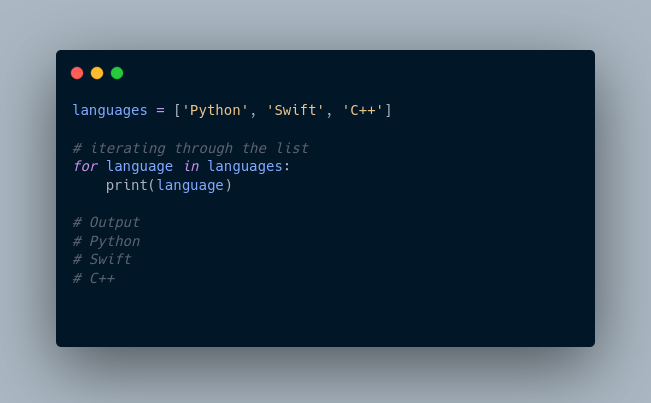
## ****Python List Methods****

Python has many useful list methods that make it really easy to work with lists.



## ****Iterating through a List****

We can use the for loop to iterate over the elements of a list. For example,

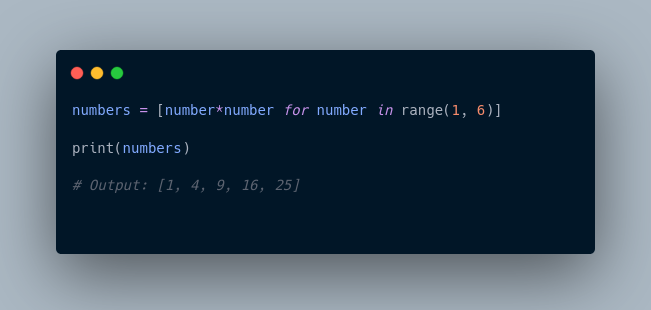


## ****Python List Comprehension****

List comprehension is a concise and elegant way to create lists.

A list comprehension consists of an expression followed by the for statement inside square brackets.

Here is an example to make a list with each item being increasing by power of **2.**

****

In the above example, we have used the list comprehension to make a list with each item being increased by power of **2.** Notice the code,



# **Tuples**

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

## ****Creating a Tuple****

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

## ****Create a Python Tuple With one Element****

In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.

We can use the type() function to know which class a variable or a value belongs to.



Here,

* ("hello") is a string so type() returns str as class of var1 i.e. <class 'str'>
* ("hello",) and "hello", both are tuples so type() returns tuple as class of var1 i.e. <class 'tuple'>

## ****Access Python Tuple Elements****

Like a list, each element of a tuple is represented by index numbers **(0, 1, ...)** where the first element is at index **0**.

We use the index number to access tuple elements. For example,

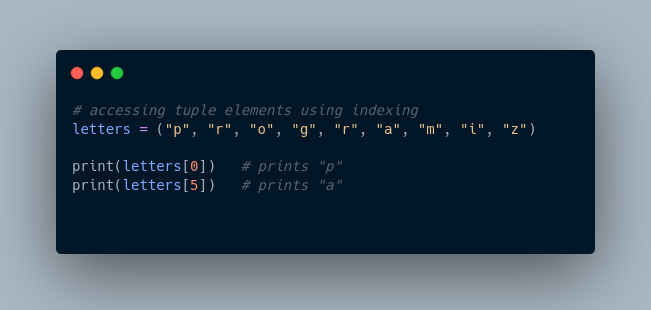
### **1. Indexing**

We can use the index operator [] to access an item in a tuple, where the index starts from 0.

So, a tuple having **6** elements will have indices from **0** to **5**. Trying to access an index outside of the tuple index range( **6,7,...** in this example) will raise an IndexError.

The index must be an integer, so we cannot use float or other types. This will result in TypeError.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.



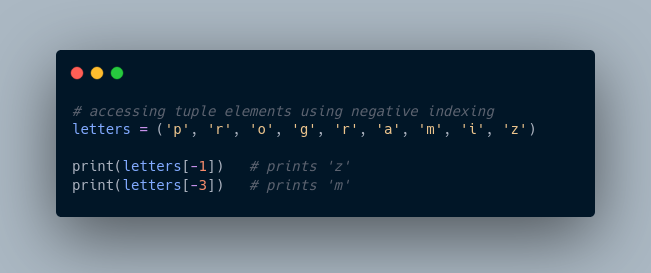
In the above example,

* letters[0] - accesses the first element
* letters[5] - accesses the sixth element

### **2. Negative Indexing**

Python allows negative indexing for its sequences.

The index of **-1** refers to the last item, **-2** to the second last item and so on. For example,



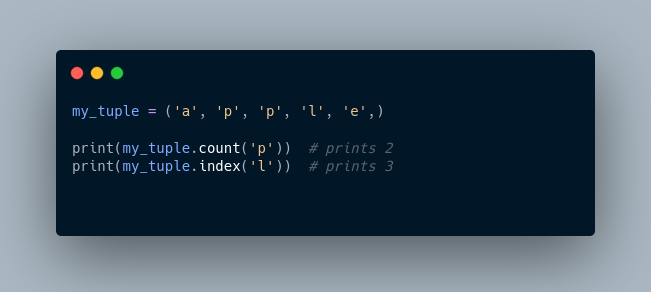
In the above example,

* letters[-1] - access last element
* letters[-3] - access third last element

## ****Python Tuple Methods****

In Python, methods that add items or remove items are not available with tuples. Only the following two methods are available.

Some examples of Python tuple methods:



Here,

* my\_tuple.count('p') - counts total number of 'p' in my\_tuple
* my\_tuple.index('l') - returns the first occurrence of 'l' in my\_tuple

**More Resources:**

1. <https://www.knowledgehut.com/tutorials/python-tutorial/python-lists-tuples>
2. <https://www.freecodecamp.org/news/python-tuple-vs-list-what-is-the-difference/>
3. <https://builtin.com/software-engineering-perspectives/python-tuples-vs-lists>
4. <https://realpython.com/python-lists-tuples/>

**>>> # A list of three numbers**

**>>> numbers = [1,2,3]**

**>>> print(numbers0**

**... ^X**

**KeyboardInterrupt**

**>>> print(numbers)**

**[1, 2, 3]**

**>>> # creating an empty list**

**>>> my\_list = []**

**>>> # list with mixed data types**

**>>> my\_list = [1, "Hello", 3.4]**

**>>> print(my\_list)**

**[1, 'Hello', 3.4]**

**>>> #list indexing**

**>>> languages = ["swift","ruby","java","python"]**

**>>> print(languages[1])**

**ruby**

**>>> #slicing - accessing a section of items from the list using the slicing operator**

**>>> languages = ["swift","ruby","java","python"]**

**>>> #access item at index 0**

**>>> print(languages[-1]) # python**

**python**

**>>> my\_list = ['p','r','o','g','r','a','m','i']**

**>>> # items from index 2 to 4**

**>>> print(my\_list[2:5])**

**['o', 'g', 'r']**

**>>> print(my\_list[5:])**

**['a', 'm', 'i']**

**>>> print(my\_list[:])**

**['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i']**

**>>> print(languages[-3])**

**ruby**

**>>>**

**Append ()- adding new values to a list**

**>>> #Add elements to a python list**

**>>> #Append()**

**>>> numbers = [2,53,35,34]**

**>>> print("Before Append:", numbers)**

**Before Append: [2, 53, 35, 34]**

**>>> #using append method**

**>>> numbers.append(304)**

**>>> print("After Append:", numbers)**

**After Append: [2, 53, 35, 34, 304]**

**>>>**

**Remove() – delete item from a list**

#using remove - to delete a list item

>>> languages = ["python","r","java","rust","c++","|"]

>>> languages.remove("|")

>>> print(languages)

['python', 'r', 'java', 'rust', 'c++']

**Iterating through a List**

languages =["java","cpp","swift"]

>>> for language in languages:

... print(language)

...

java

cpp

swift

>>>

**Python list comprehension**

#python list comprehension

>>> #example to make a list with each item being increasing by power of 2.

>>>

>>> numbers= [number\*number for number in range(1,6)]

>>> print(numbers)

[1, 4, 9, 16, 25]

>>>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Tuples**

>>> #Tuples

>>> #Tuples are immutable meaning that once theyre are created cannot be modified

≫>The index of **-1** refers to the last item, **-2** to the second last item and so on

>>> # accessing tuple elements using indexing

>>> letters =("p","r","o","g","r","a","m","i","z")

>>> print(letters[0])

p

>>> # negative indexing

>>> print(letters[-1])

z

>>>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Python tuple methods**

>>> #python tuple methods

>>> my\_tuple = ('a','p','p','l','e',)

>>> print(my\_tuple.count('p'))

2

>>> print(my\_tuple.count('l'))

1

>>> print(my\_tuple.index('l'))

3

>>>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Dictionaries**

Let's see an example,

If we want to store information about countries and their capitals, we can create a dictionary with country names as **keys** and capitals as **values**.

## Create a dictionary in Python

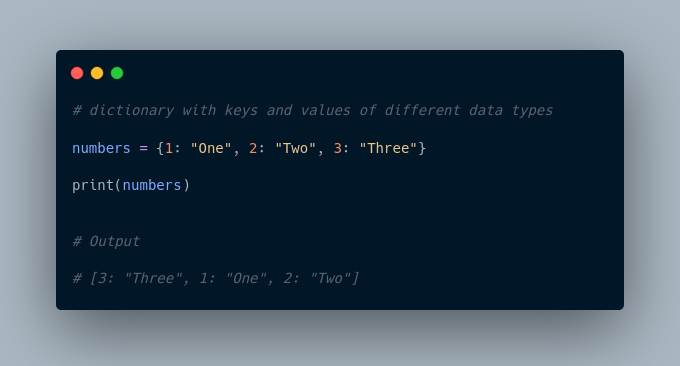
Here's how we can create a dictionary in Python.

In the above example, we have created a dictionary named capital\_city. Here,

1. **Keys** are "Nepal", "Italy", "England"
2. **Values** are "Kathmandu", "Rome", "London"

**Note**: Here, **keys** and **values** both are of string type. We can also have **keys** and **values** of different data types.

## Example 1: Python Dictionary



In the above example, we have created a dictionary named numbers. Here, **keys** are of integer type and **values** are of string type.

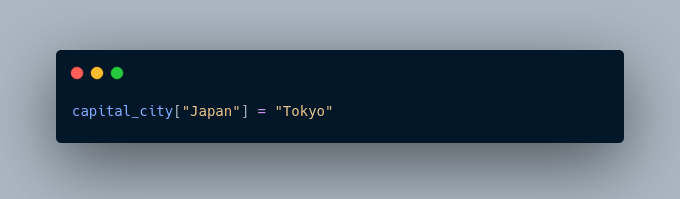
## Add Elements to a Python Dictionary

We can add elements to a dictionary using the name of the dictionary with [].

For example,



In the above example, we have created a dictionary named capital\_city. Notice the line,



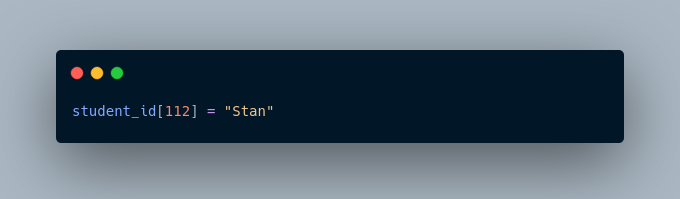
Here, we have added a new element to capital\_city with **key**: Japan and **value**: Tokyo.

## Change Value of Dictionary

We can also use [] to change the value associated with a particular key. For example,



In the above example, we have created a dictionary named student\_id. Initially, the value associated with the key 112 is "Kyle". Now, notice the line,



Here, we have changed the value associated with the key 112 to "Stan".

## Accessing Elements from Dictionary

In Python, we use the keys to access their corresponding values. For example,



Here, we have used the keys to access their corresponding values.

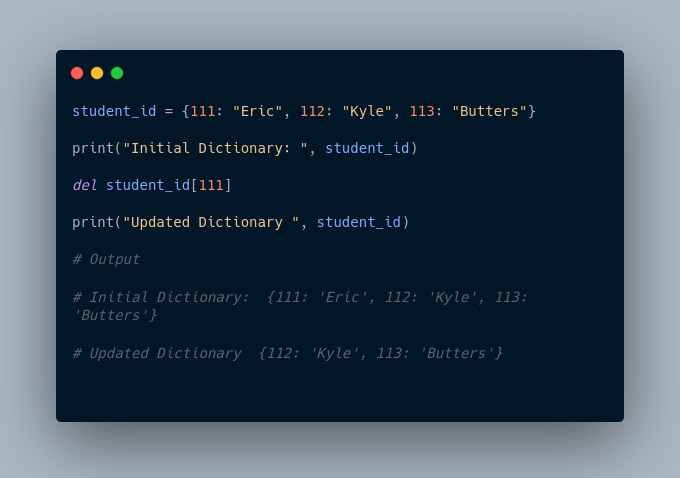
If we try to access the value of a key that doesn't exist, we'll get an error.

For example,

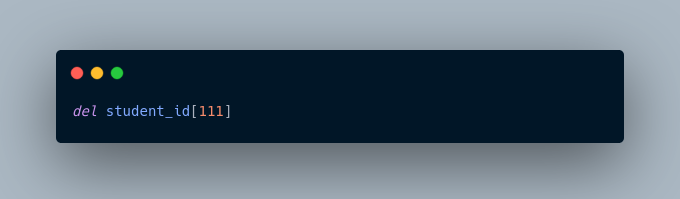


## Removing elements from Dictionary

We use the del statement to remove an element from the dictionary. For example,

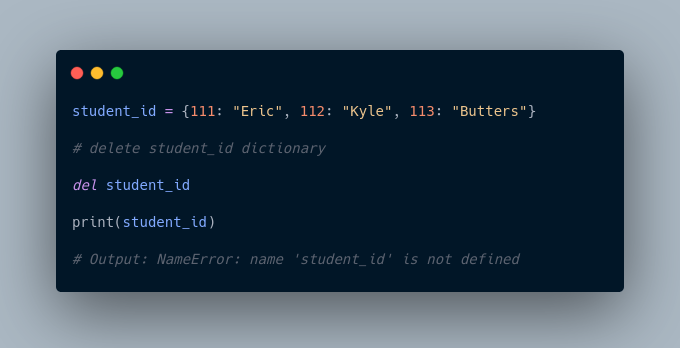


Here, we have created a dictionary named student\_id. Notice the code,



The del statement removes the element associated with the key 111.

We can also delete the whole dictionary using the del statement,



We are getting an error message because we have deleted the student\_id dictionary and student\_id doesn't exist anymore.

## Python Dictionary Methods

Methods that are available with a dictionary are tabulated below. Some of them have already been used in the above examples.

## Dictionary Membership Test

We can test if a key is in a dictionary or not using the keyword in. Notice that the membership test is only for the keys and not for the values.



## Iterating Through a Dictionary

We can iterate through each key in a dictionary using a loop.



﻿

Here, we have iterated through each **key** in the squares dictionary using the for loop.

More Resources:

1. <https://www.w3schools.com/python/python_dictionaries.asp>
2. <https://realpython.com/python-dicts/>

Python dictionary is an ordered collection (starting from **Python 3.7**) of items. It stores elements in **key/value** pairs. Here, **keys** are unique identifiers that are associated with each **value**.

**Used for storing key value pairs**

>>> #Dictionaries are to used to store key pair values

>>> capital\_city = {"Nepal": "Kathmandu", "Italy": "rome","Kenya": "Nairobi"}

>>> print(capital\_city)

{'Nepal': 'Kathmandu', 'Italy': 'rome', 'Kenya': 'Nairobi'}

>>>

1. \_\_\_\_\_\_\_\_\_\_\_\_**Keys** are "Nepal", "Italy", "England"
2. **Values** are "Kathmandu", "Rome", "London"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**#Add elemenents to a dictionary**

>>> capital\_city = {"Nepal": "Kathmandu", "England": "London"}

>>> print("Initial Dictionary: ", capital\_city)

Initial Dictionary: {'Nepal': 'Kathmandu', 'England': 'London'}

>>> capital\_city["Kenya"] = "Nairobi"

>>> print("Updated Dictionary: ", capital\_city)

Updated Dictionary: {'Nepal': 'Kathmandu', 'England': 'London', 'Kenya': 'Nairobi'}

>>>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Changing key value pairs / changing a dictionary**

# Dictionaries in python

# Changing value of dictionary

student\_id = {111: "Eric", 112:"Kyle",113:"joe"}

print("Initial dictionary: ", student\_id)

del student\_id[111]

print("updated Dictionary", student\_id)

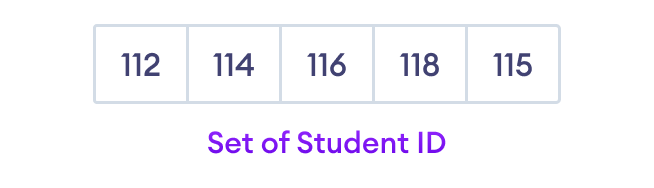
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Python sets**

A set is a collection of unique data. That is, elements of a set cannot be duplicated.

For example,

Suppose we want to store information about **student IDs**. Since **student IDs** cannot be duplicated, we can use a set.



Python Set Elements

## ****Create a Set in Python****

In Python, we create sets by placing all the elements inside curly braces {}, separated by comma.

A set can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

Let's see an example,



In the above example, we have created different types of sets by placing all the elements inside the curly braces {}.

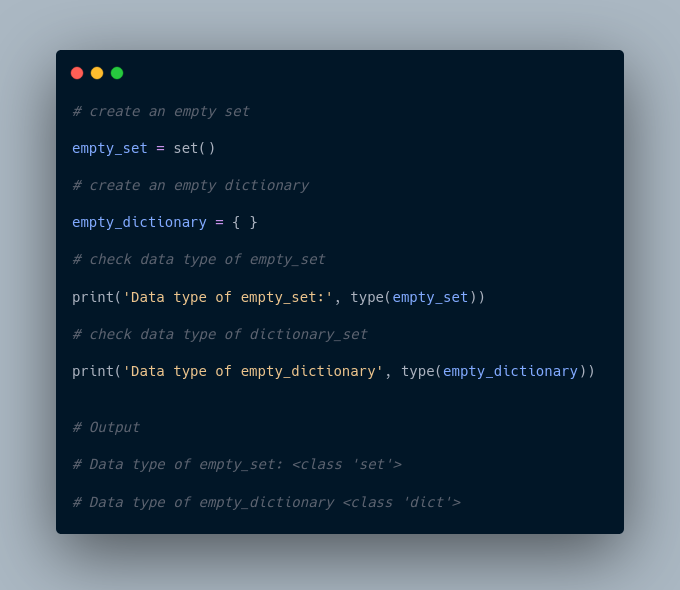
**Note:** When you run this code, you might get output in a different order. This is because the set has no particular order.

## ****Create an Empty Set in Python****

Creating an empty set is a bit tricky. Empty curly braces {} will make an empty dictionary in Python.

To make a set without any elements, we use the set() function without any argument.

For example,



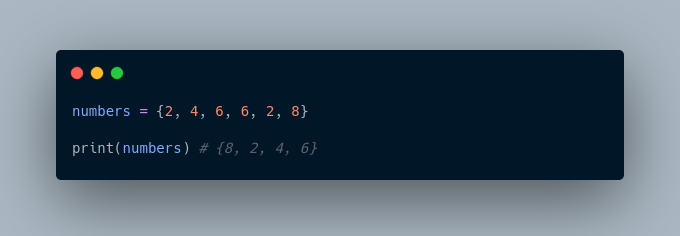
Here,

* empty\_set - an empty set created using set()
* empty\_dictionary - an empty dictionary created using {}

Finally we have used the type() function to know which class empty\_set and empty\_dictionary belong to.

## ****Duplicate Items in a Set****

Let's see what will happen if we try to include duplicate items in a set.



Here, we can see there are no duplicate items in the set as a set cannot contain duplicates.

## ****Add and Update Set Items in Python****

Sets are mutable. However, since they are unordered, indexing has no meaning.

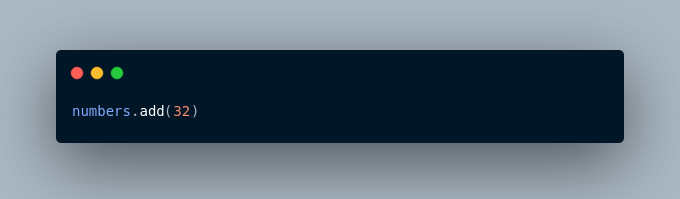
We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

### **Add Items to a Set in Python**

In Python, we use the add() method to add an item to a set. For example,



In the above example, we have created a set named numbers. Notice the line,

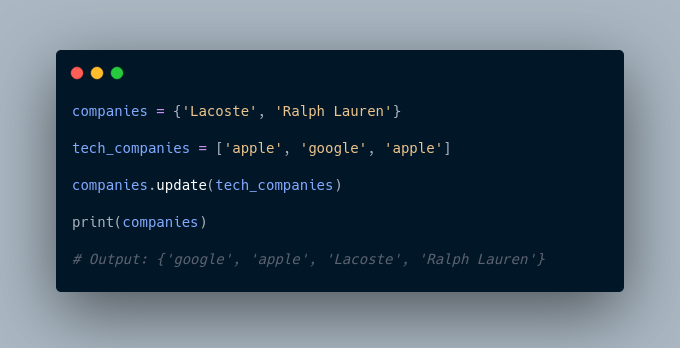


Here, add() adds **32** to our set.

### **Update Python Set**

The update() method is used to update the set with items other collection types (lists, tuples, sets, etc).

For example,

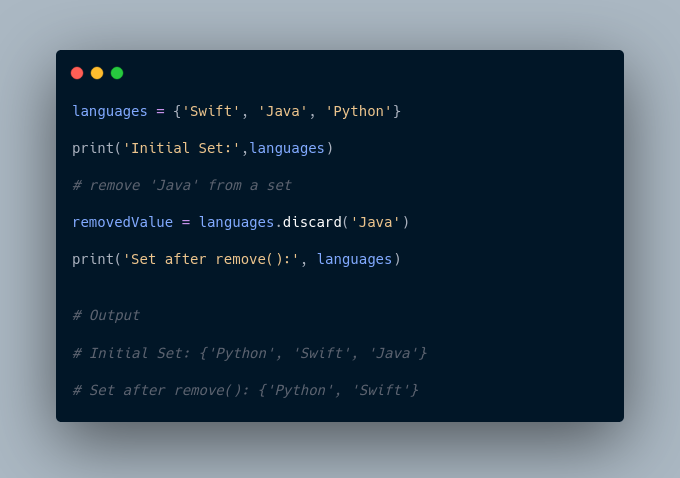


Here, all the unique elements of tech\_companies are added to the company's set.

## ****Remove an Element from a Set****

We use the discard() method to remove the specified element from a set.

For example,



Here, we have used the discard() method to remove 'Java' from the languages set.

## ****Built-in Functions with Set****

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with sets to perform different tasks.



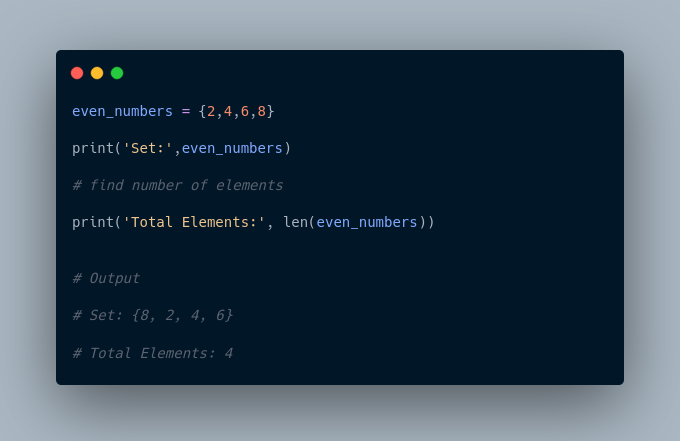
## ****Iterate Over a Set in Python****

****

## ****Find Number of Set Elements****

We can use the len() method to find the number of elements present in a Set.

For example,



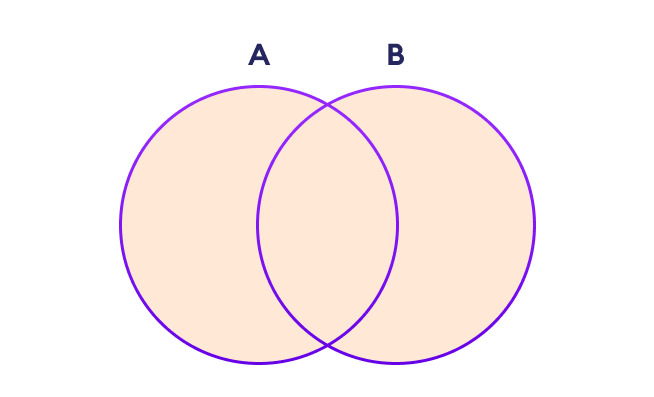
Here, we have used the len() method to find the number of elements present in a Set.

## ****Python Set Operations****

Python Set provides different built-in methods to perform mathematical set operations like union, intersection, subtraction, and symmetric difference.

## ****Union of Two Sets****

The union of two sets **A** and **B** include all the elements of set **A** and **B**.



Set Union in Python

We use the | operator or the union() method to perform the set union operation.

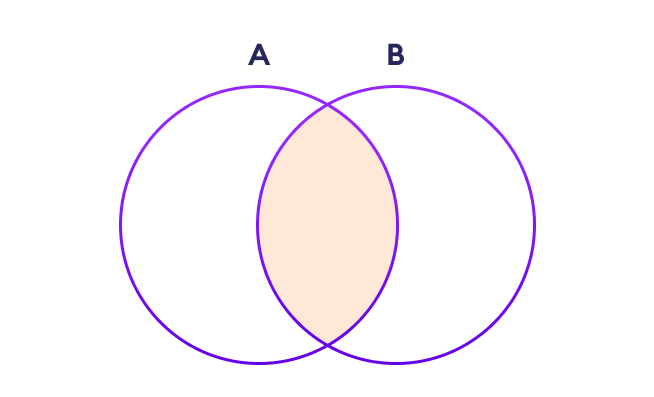
For example,



**Note**: A|B and union() is equivalent to A ⋃ B set operation.

## ****Set Intersection****

The intersection of two sets **A** and **B** include the common elements between set **A** and **B**.



Set Intersection in Python

In Python, we use the & operator or the intersection() method to perform the set intersection operation.

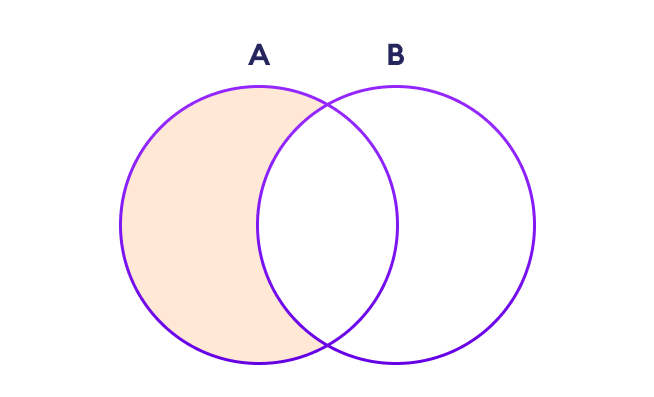
For example,



**Note**: A&B and intersection() is equivalent to A ⋂ B set operation.

## ****Difference between Two Sets****

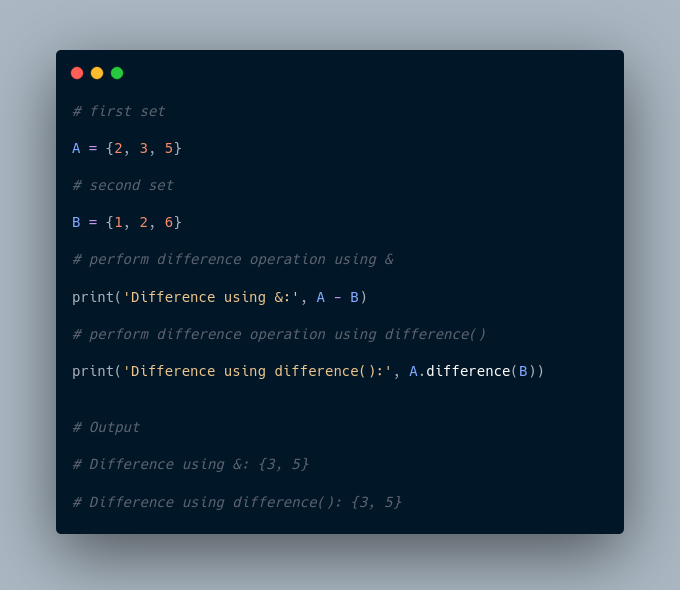
The difference between two sets **A** and **B** include elements of set **A** that are not present on set **B**.



Set Difference in Python

We use the - operator or the difference() method to perform the difference between two sets.

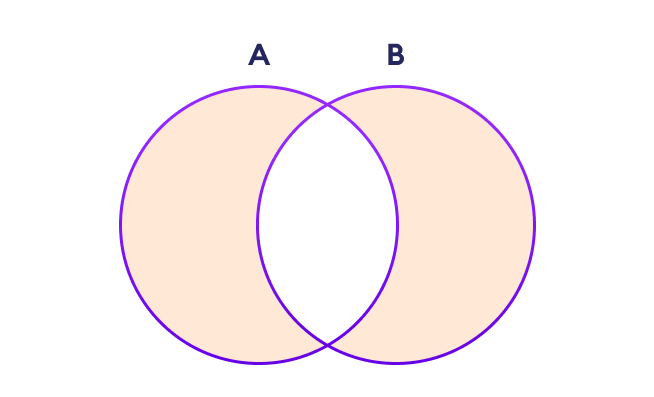
For example,



**Note**: A - B and A.difference(B) is equivalent to A - B set operation.

## ****Set Symmetric Difference****

The symmetric difference between two sets **A** and **B** includes all elements of **A** and **B** without the common elements.



Set Symmetric Difference in Python

In Python, we use the ^ operator or the symmetric\_difference() method to perform symmetric difference between two sets.

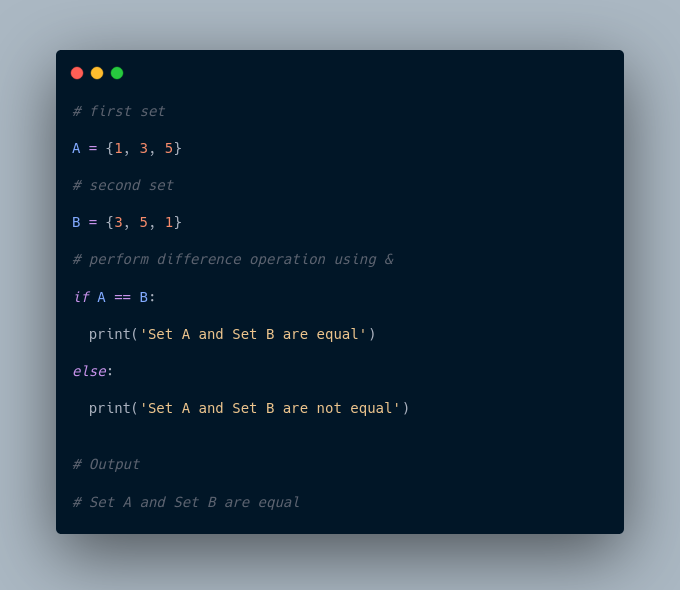
For example,



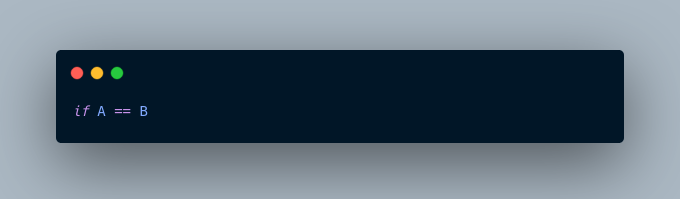
## ****Check if two sets are equal****

We can use the == operator to check whether two sets are equal or not.

For example,



In the above example, A and B have the same elements, so the condition

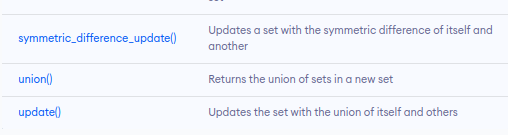


evaluates to True. Hence, the statement print('Set A and Set B are equal') inside the if is executed.

## ****Other Python Set Methods****

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with the set objects:





More Resources:

1. <https://www.geeksforgeeks.org/sets-in-python/>
2. <https://realpython.com/python-sets/>

**Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32**

**Type "help", "copyright", "credits" or "license" for more information.**

**>>> #set**

**>>> # A set is a collection of unique date. The elements of a set cannot be duplicated**

**>>> #Eg student id cannot be duplicated**

**>>> # A set can have any number of items and they may be of different types(integer, float, tuple, string) but it cannot have a mutable elements like lists, sets or ductionaries**

**>>> #eg**

**>>> #sets are created by placing all the elements inside curly braces {}, separated by a comma**

**>>> #create a set of integer type**

**>>> std\_id = {11,12,13,14,15,16}**

**>>> print('Student ID:', std\_id)**

**Student ID: {16, 11, 12, 13, 14, 15}**

**>>> #create a set of string type**

**>>> vowel\_letters ={'a','e','i','o','u'}**

**>>> print('Vowel Letters:', vowel\_letters)**

**Vowel Letters: {'u', 'o', 'i', 'e', 'a'}**

**>>> #create a set of mixed data types**

**>>> mixed\_set ={"Hi", "I love coding wooow",1,2,3,"Byeeee"}**

**>>> print('Set of mixed data types: ', mixed\_set)**

**Set of mixed data types: {1, 'I love coding wooow', 3, 2, 'Byeeee', 'Hi'}**

**>>> #The output displayed has no order since sets do not have an order**

**>>> #To create an empty set we use set() function without any argument**

**>>> #create empty set**

**>>> empty\_set = set ()**

**>>> #create empty dictionary**

**>>> empty\_dictionary ={ }**

**>>> #check data type of empty set**

**>>> print('Data type of empty\_set:', type(empty\_set))**

**Data type of empty\_set: <class 'set'>**

**>>> #check data type of dictionary set**

**>>> print('Data type of empty\_dictionary:', type(empty\_dictionary))**

**Data type of empty\_dictionary: <class 'dict'>**

**>>> # create duplicate items. nb sets dont have duplicates**

**>>> number =[1,3,4,2,2,1}**

**File "<stdin>", line 1**

**number =[1,3,4,2,2,1}**

**^**

**SyntaxError: closing parenthesis '}' does not match opening parenthesis '['**

**>>> numbers ={1,2,3,2,3,2,32,2}**

**>>> print(numbers)**

**{32, 1, 2, 3}**

**>>> #Add items to a set in python**

**>>> numbers={1,2,43,2,3,4}**

**>>> numbers.add(100)**

**>>> print('Updated set:', numbers)**

**Updated set: {1, 2, 3, 4, 100, 43}**

**>>> #Update python set**

**>>> # we use update() method**

**>>> companies = {'lacoste', 'Ralph Lauren'}**

**>>> tech\_companies = ['apple','google','apple']**

**>>> companies.update(tech\_companies)**

**>>> print(companies)**

**{'Ralph Lauren', 'apple', 'google', 'lacoste'}**

**>>> #Remove an element from a set**

**>>> languages ={'java', 'swift','python','r','scala'}**

**>>> print('Initial set:',languages)**

**Initial set: {'r', 'java', 'swift', 'python', 'scala'}**

**>>> #we use discard() method to remove an element from a set**

**>>> remove\_element = languages.discard('java')**

**>>> print('Set after remove():', languages)**

**Set after remove(): {'r', 'swift', 'python', 'scala'}**

**>>> #Iterate over a set**

**>>> fruits = {"apple","mangoes","peach","dates"}**

**>>> #for loop to access each fruit**

**>>> for fruit in fruits:**

**... print(fruit)**

**...**

**apple**

**mangoes**

**peach**

**dates**

**>>> #Find number of set elements**

**>>> even\_numbers = {2,4,6,8}**

**>>> print('set:', even\_numbers)**

**set: {8, 2, 4, 6}**

**>>> #find number of elements**

**>>> print('Total element:', len(even\_numbers))**

**Total element: 4**

**>>> #Python set operations**

**>>> #Python set provides different built-in methods to perform mathematical set operations like union, intersection, subtraction and symmetric difference**

**>>> #set union.. we use | operator or the union() method to perform the set union operation**

**>>> #first ser**

**>>> A = {1,3,5}**

**>>> #second set**

**>>> B={0,2,4}**

**>>> #perform union operation using |**

**>>> print('union using |:', A | B)**

**union using |: {0, 1, 2, 3, 4, 5}**

**>>> #perform union using union()**

**>>> print('Union method using Union():',A.union(B))**

**Union method using Union(): {0, 1, 2, 3, 4, 5}**

**>>> #Set intersection - the intersection of two sets A and B include the common elements between set A and B**

**>>> #First set**

**>>> A={1,3,5}**

**>>> #second set**

**>>> B ={1,2,3}**

**>>> #perform set intersection using ambassand operant (&)**

**>>> print('intersection using &:', A & B)**

**intersection using &: {1, 3}**

**>>> # using intersection method**

**>>> print('intersection using interesection():',A.intersection(B))**

**intersection using interesection(): {1, 3}**

**>>> #Set symmetric Difference in python**

**>>> #We use ^ operator or the the symmetric\_difference() method**

**>>> #first set**

**>>> A={2,3,5}**

**>>> #second set**

**>>> B={1,2,6}**

**>>> #perform the difference using the ^**

**>>> print('Difference using^:', A ^ B)**

**Difference using^: {1, 3, 5, 6}**

**>>> # Using difference function**

**>>> print('Using symmetric\_difference():', A.symmetric\_difference(B))**

**Using symmetric\_difference(): {1, 3, 5, 6}**

**>>> #check if two symmetric sets are equal**

**>>> #first set**

**>>> A={1,3,5}**

**>>> #second set**

**>>> B={3,5,1}**

**>>> #perform difference operation using ==**

**>>> if A == B:**

**... print('Set A and B are equal')**

**... else:**

**... print('Set A and B are not equal')**

**...**

**Set A and B are equal**

**>>>**

**Python String and String Methods**

In computer programming, a string is a sequence of characters.

For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

## ****Object Oriented Programming****

Object-Oriented Programming (OOP) is a programming paradigm in computer science that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.

## ****Building blocks of OOP****

* Classes - a blueprint of an object
* Objects - an instance of a class
* Methods - methods represent behaviors
* Attributes - information to be stored in a class about an object

## ****Four Principles of OOP****

## The four pillars of object-oriented programming are:

## ****Inheritance****: child classes inherit data and behaviors from the parent class

## ****Encapsulation****: containing information in an object, exposing only selected information

## ****Abstraction****: only exposing high-level public methods for accessing an object

## ****Polymorphism****: many methods can do the same task

## ****What are the benefits of OOP?****

## Modularity.

* Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.

## Reusability.

* Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.

## Productivity.

* Programmers can construct new programs quickly through the use of multiple libraries and reusable code.

## Easily upgradable and scalable.

* Programmers can implement system functionalities independently.

## Interface descriptions.

* Descriptions of external systems are simple, due to message-passing techniques that are used for object communication.

## Security.

* Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and [internet protocols](https://www.techtarget.com/searchunifiedcommunications/definition/Internet-Protocol) are protected.

## ****Flexibility****.

* Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface.

**Class Attributes**

Class attributes are the variables defined directly in the class that are shared by all objects of the class. Class attributes can be accessed using the class name as well as using the objects.

class Person:

    name = 'Skinny’ #Class attribute

Above, the name is a class attribute defined inside a class Person. The value of the name will remain the same for all the objects unless modified explicitly.

**Accessing class attributes**

>>> Person.name

'Skinny'

>>> details = Person()

>>> details.name

'Skinny'

**Constructor**

In Python, the constructor method is invoked automatically whenever a new object of a class is instantiated, same as constructors in C++ or Java.

The constructor must have a special name \_\_init\_\_() and a special parameter called self.

All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.

The constructor in Python is used to define the attributes of an instance and assign values to them.

NB: The first parameter of each method in a class must be the self, which refers to the calling object. However, you can give any name to the first parameter, not necessarily self.

Example of how to define a constructor:

class Person:

    def \_\_init\_\_(self): # constructor method

        print('Constructor invoked')

Now, whenever you create an object of the Person class, the \_\_init\_\_() constructor method will be called, as shown below.

>>>details1 = Person()

Constructor invoked

>>>details2 = Person()

Constructor invoked

**Instance Attributes**

Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.

The following example defines instance attributes name and age in the constructor.

class Person:

    nationality = 'Ethiopian' # class attribute

    def \_\_init\_\_(self): # constructor

        self.name = '' # instance attribute

        self.age = 0 # instance attribute

To access an instance variable, we use dot notation:

* [instance name].[attribute name], as shown below

>>> p1 = Person()

>>> p1.name

''

>>> p1.age

0

You can set the value of attributes using the dot notation, as shown below.

>>> p1 = Person()

>>> p1.name = "Mutemi" # assign value to instance attribute

>>> p1.age = 65    # assign value to instance attribute

>>> p1.name     # access instance attribute value

Mutemi

>>> std.age     # access value to instance attribute

65

The best practice is to always specify the values of instance attributes through the constructor.

***Setting Attribute Values***

The following constructor includes the name and age parameters, other than the self parameter.

class Person:

# name & age parameters passed in constructor

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

***Passing Instance Attribute Values in Constructor***

Now, you can specify the values while creating an instance, as shown below.

>>> p1 = Person('Mutemi', 65)

>>> p1.name

'Mutemi'

>>> p1.age

65

***Setting Default Values of Instance Attributes***

Also, you can set default values to instance attributes. By doing this, if the values are not provided when creating an object, the default values will be assigned.

Lets assign name=”mkuu” and age=101

class Person:

    def \_\_init\_\_(self, name="mkuu", age=101)

        self.name=name

        self.age=age

***Instance Attribute Default Value***

Now, you can create an object with default values, as shown below

>>> p1 = Person()

>>> p1.name

'Guest'

>>> p1.age

65

**Class Methods**

A python function in a class is called class method. Methods are defined using the def keyword.

Each method must have a self as the first parameter, which refers to calling the instance.

* Self is just a conventional name for the first argument of a method in the class.
* A method defined as mymethod(self, a, b) should be called as x.mymethod(a, b) for the object x of the class.

Example: here we have a method named displayInfo

class Person:

    def displayInfo(self): # class method

        print('Personal Information')

The above class method can be called as a normal function, as shown below.

>>> p1 = Person()

>>> p1.displayInfo()

'Personal Information'

Let's combine our knowledge so far for class constructors and methods to access instance attributes using self parameter.

class Person:

    def \_\_init\_\_(self, name, age): # class constructor

        self.name = name

        self.age = age

    def displayInfo(self): # class method

        print('Person Name: ', self.name,', Age: ', self.age)

*Calling a Method*

Let's call/Invoke the displayInfo method as shown below:

>>> p1 = Person('Mutemi', 65)

>>> p1.displayInfo()

Person Name: Mutemi , Age: 65

## What is Inheritance?

The process of inheriting the properties of the parent class into a child class is called inheritance. The existing class is called a base class or parent class and the new class is called a subclass or child class or derived class.

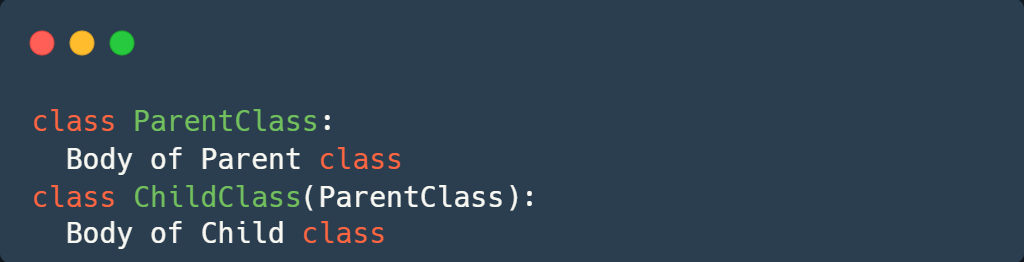
 Inheritance promotes code reusability, which is considered one of the best industrial coding practices as it makes the codebase modular.

In inheritance, the child class acquires all the data members, properties, and functions from the parent class. Also, a child class can also provide its specific implementation to the methods of the parent class.

Example:

In the real world, a Car is a subclass/Child of a Vehicle class(Parent). We can create a Car by inheriting the properties of a Vehicle such as Wheels, Colors, Fuel tank, engine, and add extra properties in Car as required.

Syntax

****

Types of inheritance in Python

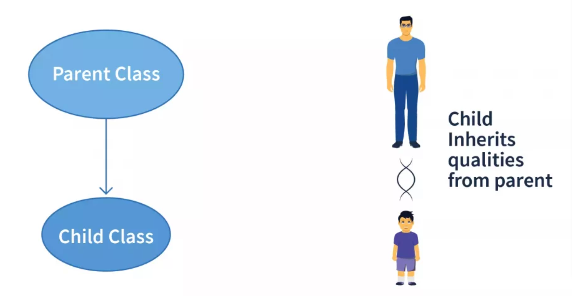
There are 5 different types of inheritance. This is based on the number of child and parent classes involved.

Let's explore them here:

1. Single inheritance
2. Multiple Inheritance
3. Multilevel inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

## ****Single Inheritance****

A child class inherits from a single parent class

****

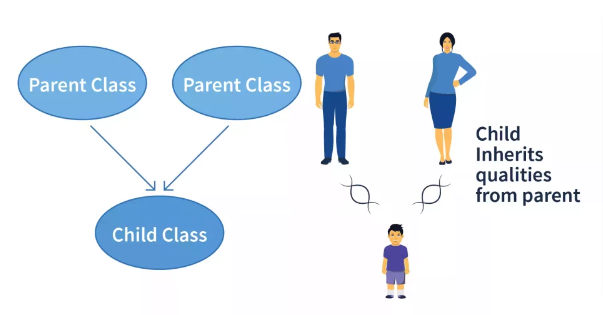
Example

Let’s create one parent class called Vehicle and one child class called Car to implement single inheritance.

****

## ****Multiple Inheritance****

Here, a one Child class inherits from multiple parent classes. It means the child class has access to all the parent classes' methods and attributes.

****

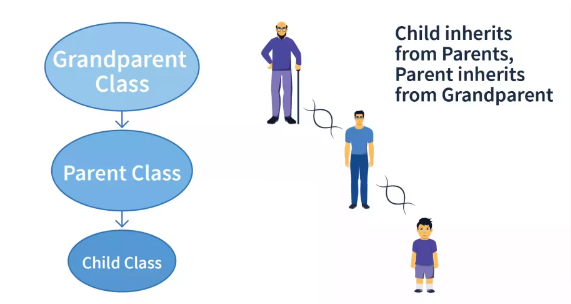
Example:

Let's create two parent 2 Parent classes Person and Company . Then create a Child class Employee. Which inherits from the two parent classes.



## ****Multilevel inheritance****

In multilevel inheritance, a class inherits from a child class or derived class. Suppose three classes A, B, C. A is the superclass, B is the child class of A, C is the child class of B. In other words, we can say a chain of classes is called multilevel inheritance.



Example:

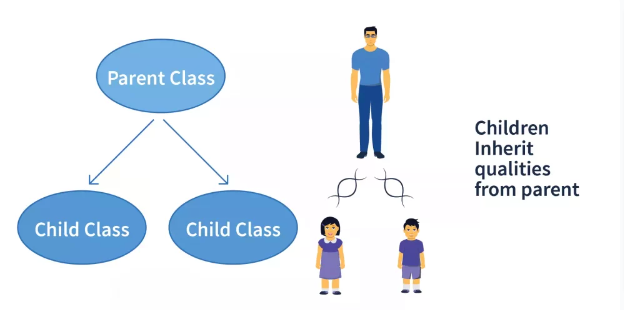
Let’s create 3 classes namely, Vehicle, Car, SportsCar. Here, Vehicle is the superclass, Car is a child class of Vehicle, SportsCar is a child of Car.

This is a good example of chaining in class/ Multilevel inheritance



## ****Hierarchical Inheritance****

Hierarchical Inheritance is the right opposite of multiple inheritance. It means that there are multiple derived child classes from a single parent class.



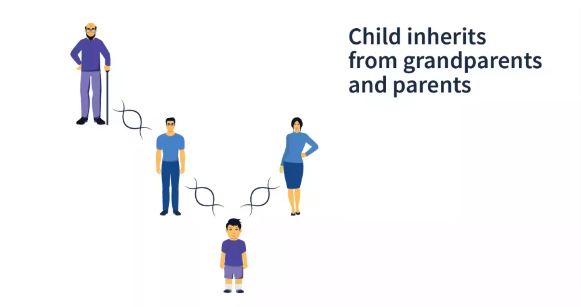
Example:

Let’s create ‘Vehicle’ as a parent class and two child classes ‘Car’ and ‘Truck’ as a parent class.



## ****Hybrid Inheritance****

Hybrid Inheritance is the mixture of two or more different types of inheritance. Here we can have many relationships between parent and child classes with multiple levels.

****

Example:

Let’s implement hierarchical and multiple inheritance combinations.

Create a Parent class Vehicle and two child classes Car and Truck (hierarchical inheritance)

Create another class SportCar that inherits from two Parent classes named Car and Vehicle (Multiple inheritance)

# hybrid inheritance example

****

## ****Python super() function****

Python also has a super() function that will make the child class inherit all the methods and properties from its parent

Example:

In the example below, Let’s create a parent class Company and child class Employee. In Employee class, we call the parent class method by using a super() function.

****

## ****issubclass()****

In Python, we can verify whether a particular class is a subclass of another class. For this purpose, we can use Python built-in function issubclass(). This function returns True if the given class is the subclass of the specified class. Otherwise, it returns False.

Syntax

issubclass(class, classinfo)

Where,

* **class: class to be checked.**
* **classinfo: a class, type, or a tuple of classes or data types.**

Example

****

**More Resources**

Python-Polymorphism: <https://pynative.com/python-polymorphism/>

Python-Encapsulation: <https://pynative.com/python-encapsulation/>

# **Access modifiers**

## Access modifiers play an important role in securing the data from unauthorized access and preventing any data exploitation.

## Using the underscore (\_), we can control access to the data inside the Python classes. Python Class has three types of access modifiers:

* Public Access Modifier
* Private Access Modifier
* Protected Access Modifier

# **Public Members**

## The name "Public" says all about this access modifier: the variables and methods declared inside the specific Python class can be accessed by that class and any Python class outside that specific class.

## Public methods are accessible outside the class and with the help of objects the public methods can be invoked inside the class.

## All members in a Python class are public by default. Any member can be accessed from outside the class environment.

Example:



You can access the Student class's attributes and also modify their values, as shown above.

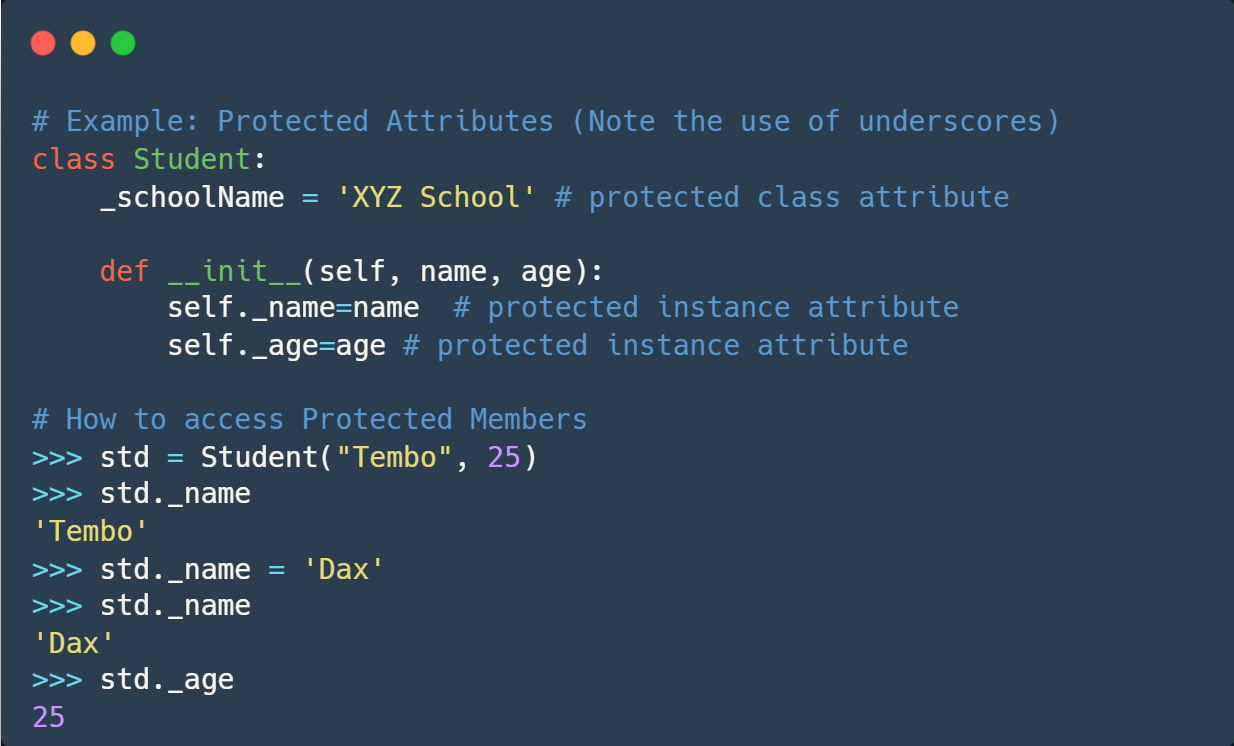
# **Protected Members**

Protected members of a class are accessible from within the class and are also available to its subclasses. No other environment is permitted access to it.

This enables specific resources of the parent class to be inherited by the child class.

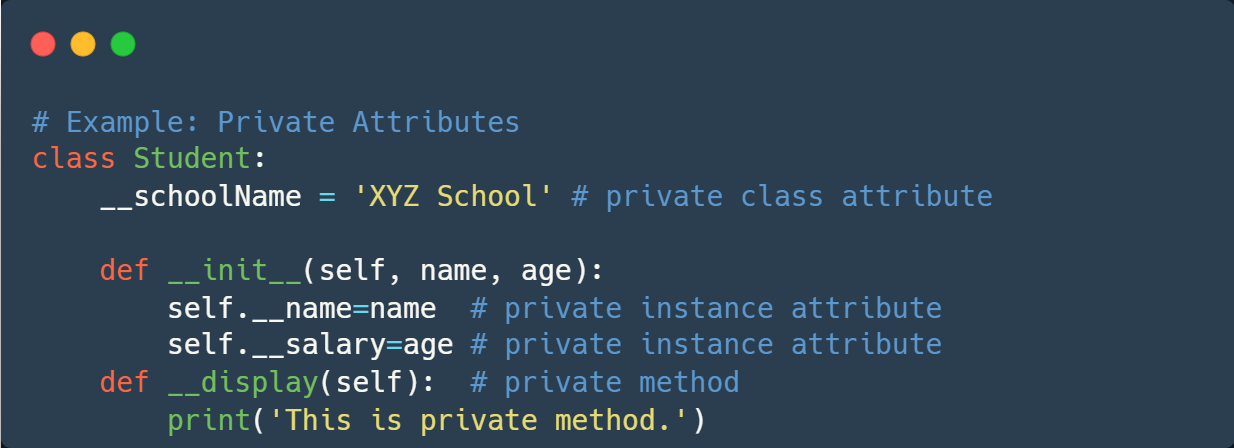
Conventionally, to make an instance variable protected you add a prefix \_ (single underscore) to it. This effectively prevents it from being accessed unless it is from within a sub-class.

Example:



# **Private Members**

You can make an instance variable or method private by using the double underscore \_\_, like:



Python performs name mangling of private variables. Every member with a double underscore will be changed to \_object.\_class\_\_variable.So, it can still be accessed from outside the class, but the practice should be refrained.

