Python Classes and Objects An object is simply a collection of data (variables) and methods (functions). Similarly, a class is a blueprint for that object.Python Classes A class is considered a blueprint of objects. We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions, we build the house; the house is the object. Since many houses can be made from the same description, we can create many objects from a class.Define Python Class class ClassName: # class definition class Bike: name = "" gear = 0Here, Bike - the name of the class name/gear - variables inside the class with default values "" and 0 respectively.Python Objects An object is called an instance of a class. Suppose Bike is a class then we can create objects like bike1, bike2, etc from the class. Here's the syntax to create an object.objectName = ClassName()

In [1]:
```
# define a class
class Bike:
    name = ""
    gear = 0

# create object of class
bike1 = Bike()

# access attributes and assign new values
bike1.gear = 11
bike1.name = "Mountain Bike"

print(f"Name: {bike1.name}, Gears: {bike1.gear} ")
```

```
Name: Mountain Bike, Gears: 11
```

In [2]:
```
#Create Multiple Objects of Python Class
# define a class
class Employee:
    # define a property
    employee_id = 0

# create two objects of the Employee class
employee1 = Employee()
employee2 = Employee()

# access property using employee1
employee1.employeeID = 1001
print(f"Employee ID: {employee1.employeeID}")

# access properties using employee2
employee2.employeeID = 1002
print(f"Employee ID: {employee2.employeeID}")
```

```
Employee ID: 1001
Employee ID: 1002
```

In [ ]:
```
# create a class
class Room:
    length = 0.0
    breadth = 0.0

    # method to calculate area
    def calculate_area(self):
        print("Area of Room =", self.length * self.breadth)

# create object of Room class
study_room = Room()
```

```
# assign values to all the properties
study_room.length = 42.5
study_room.breadth = 30.8

# access method inside class
study_room.calculate_area()
```

Python standard library The Python Standard Library contains the exact syntax, semantics, and tokens of Python. It contains built-in modules that provide access to basic system functionality like I/O and some other core modules. Most of the Python Libraries are written in the C programming language. The Python standard library consists of more than 200 core modules. All these work together to make Python a high-level programming language. Python Standard Library plays a very important role. Without it, the programmers can't have access to the functionalities of Python. But other than this, there are several other libraries in Python that make a programmer's life easier. Let's have a look at some of the commonly used libraries: TensorFlow: This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics. Matplotlib: This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc. Pandas: Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc. Numpy: The name "Numpy" stands for "Numerical Python". It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library. SciPy: The name "SciPy" stands for "Scientific Python". It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers. Scrapy: It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data. Scikit-learn: It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy. PyGame: This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with Python programming language. PyTorch: PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks. PyBrain: The name "PyBrain" stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that's why is really helpful for developers that are new in research fields.

In [1]:
```
# Importing math library
import math

A = 16
print(math.sqrt(A))
```

```
4.0
```

In [2]:
```
# Importing specific items
from math import sqrt, sin

A = 16
B = 3.14
print(sqrt(A))
print(sin(B))
```

```
4.0
0.0015926529164868282
```

NumPy: For numerical computations. Pandas: For data manipulation and analysis. Matplotlib: For data visualization. SciPy: For scientific and technical computing. scikit-learn: For machine learning. TensorFlow: For deep learning. Keras: For building and training neural networks. Requests: For making HTTP requests. BeautifulSoup: For web scraping. Django: For web development.

In [3]:
```
# Python program for
# Creation of Arrays
import numpy as np
```

```
# Creating a rank 1 Array
arr = np.array([1, 2, 3])
print("Array with Rank 1: \n",arr)

# Creating a rank 2 Array
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print("Array with Rank 2: \n", arr)

# Creating an array from tuple
arr = np.array((1, 3, 2))
print("\nArray created using "
      "passed tuple:\n", arr)
```

```
Array with Rank 1:
 [1 2 3]
Array with Rank 2:
 [[1 2 3]
 [4 5 6]]

Array created using passed tuple:
 [1 3 2]
```

In [5]:
```
import pandas as pd
import numpy as np

# Creating empty series
ser = pd.Series()
print("Pandas Series: ", ser)

# simple array
data = np.array(['g', 'e', 'e', 'k', 's'])

ser = pd.Series(data)
print("Pandas Series:\n", ser)
```

```
Pandas Series:  Series([], dtype: object)
Pandas Series:
 0    g
1    e
2    e
3    k
4    s
dtype: object
```

If pandas library is not installed then use "pip install pandas" into cmd.

In [22]:
```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

print(r)
```

In [23]:
```
import numpy
numpy.version.version
```

Out[23]: '1.22.4'

In [25]:
```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[25], line 1
----> 1 import matplotlib.pyplot as plt
      2 from scipy import stats
      4 x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\__init_
_.py:264
    259         if parse_version(module.__version__) < parse_version(minver):
    260             raise ImportError(f"Matplotlib requires {modname}>={minver}; "
    261                               f"you have {module.__version__}")
--> 264 _check_versions()
    267 # The decorator ensures this always returns the same handler (and it is only
    268 # attached once).
    269 @functools.cache
    270 def _ensure_handler():

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\__init_
_.py:260, in _check_versions()
    258 module = importlib.import_module(modname)
    259 if parse_version(module.__version__) < parse_version(minver):
--> 260     raise ImportError(f"Matplotlib requires {modname}>={minver}; "
    261                       f"you have {module.__version__}")

ImportError: Matplotlib requires numpy>=1.23; you have 1.22.4
```

In [ ]:

For loop assignmentsfor <loop_variable> in range(<start>, <stop>, <step>): As you can see, the range function has three parameters: start: where the sequence of integers will start. By default, it's 0. stop: where the sequence of integers will stop (without including this value). step: the value that will be added to each element to get the next element in the sequence. By default, it's 1

In [ ]:
```
for i in range(5):
    print(i)
```

In [6]:
```
for j in range(1,11):
    print(j * 2)
```

```
2
4
6
8
10
12
14
16
18
20
```

Sanjivani SanjivaniSanjivani SanjivaniSanjivaniSanjivani SanjivaniSanjivaniSanjivaniSanjivani SanjivaniSanjivaniSanjivaniSanjivaniSanjivani SanjivaniSanjivaniSanjivaniSanjivaniSanjivaniSanjivani SanjivaniSanjivaniSanjivaniSanjivaniSanjivaniSanjivaniSanjivani

In [ ]:
```
for i in range(2, 10):
    print(i)
```

PythonPython PythonPythonPython PythonPythonPythonPython

In [8]:
```
#Code
for i in range (2,5):
    print ("python" * i)
```

```
pythonpython
pythonpythonpython
pythonpythonpythonpython
```

In [ ]:
```
for i in range(3, 16, 2): for (i=3;i<16;i+2)
    print(i)
```

In [ ]:
```
for j in range(10, 5, -1):
    print(j)
```

In [ ]:
```

```

File Handling in Python

File handling refers to the process of performing operations on a file such as creating, opening, reading, writing and closing it, through a programming interface. It involves managing the data flow between the program and the file system on the storage device, ensuring that data is handled safely and efficiently.

Open the file and read its contents

with open('sanjivani.txt', 'r') as file:

File Modes in Python

When opening a file, we must specify the mode we want to which specifies what we want to do with the file. Here's a table of the different modes available:

Mode	Description	Behavior
r	Read-only mode.	Opens the file for reading. File must exist; otherwise, it raises an error.
rb	Read-only in binary mode.	Opens the file for reading binary data. File must exist; otherwise, it raises an error.
r+	Read and write mode.	Opens the file for both reading and writing. File must exist; otherwise, it raises an error.
rb+	Read and write in binary mode.	Opens the file for both reading and writing binary data. File must exist; otherwise, it raises an error.
w	Write mode.	Opens the file for writing. Creates a new file or truncates the existing file.
wb	Write in binary mode.	Opens the file for writing binary data. Creates a new file or truncates the existing file.
w+	Write and read mode.	Opens the file for both writing and reading. Creates a new file or truncates the existing file.

Mode	Description	Behavior
wb+	Write and read in binary mode.	Opens the file for both writing and reading binary data. Creates a new file or truncates the existing file.
a	Append mode.	Opens the file for appending data. Creates a new file if it doesn't exist.
ab	Append in binary mode.	Opens the file for appending binary data. Creates a new file if it doesn't exist.
a+	Append and read mode.	Opens the file for appending and reading. Creates a new file if it doesn't exist.
ab+	Append and read in binary mode.	Opens the file for appending and reading binary data. Creates a new file if it doesn't exist.
x	Exclusive creation mode.	Creates a new file. Raises an error if the file already exists.
xb	Exclusive creation in binary mode.	Creates a new binary file. Raises an error if the file already exists.
x+	Exclusive creation with read and write mode.	Creates a new file for reading and writing. Raises an error if the file exists.
xb+	Exclusive creation with read and write in binary mode.	Creates a new binary file for reading and writing. Raises an error if the file exists.

Reading a File

Reading a file can be achieved by **file.read()** which reads the entire content of the file. After reading the file we can close the file using file.close() which closes the file after reading it, which is necessary to free up system resources.

Example: Reading a File in Read Mode (r)

```
file = open("sanjivani.txt", "r")
content = file.read()
print(content)
file.close()
```

In [13]:
```
f = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "r")
print(f.read())
```

```
Hello! Welcome to Sanjivani.txt
This file is for testing purposes.
Good Luck!
```

In [16]:
```
f = open("D:\\Cyber.txt", "r")
print(f.read())
```

In [17]:
```
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "r")
content = file.read()
print(content)
file.close()
```

```
Hello! Welcome to Sanjivani.txt
This file is for testing purposes.
Good Luck!
```

Writing to a File Writing to a file is done using file.write() which writes the specified string to the file. If the file exists, its content is erased. If it doesn't exist, a new file is created.

In [ ]:
```
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "w")
file.write("Hello, World!")
file.close()
```

Writing to a File in Append Mode (a) It is done using file.write() which adds the specified string to the end of the file without erasing its existing content.

In [ ]:
```
# Python code to illustrate append() mode
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", 'a')
file.write("This will add this line")
file.close()
```

Closing a File Closing a file is essential to ensure that all resources used by the file are properly released. file.close() method closes the file and ensures that any changes made to the file are saved.

In [ ]:
```
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "r")
# Perform file operations
file.close()
```

Advantages of File Handling in Python Versatility : File handling in Python allows us to perform a wide range of operations, such as creating, reading, writing, appending, renaming and deleting files. Flexibility : File handling in Python is highly flexible, as it allows us to work with different file types (e.g. text files, binary files, CSV files , etc.) and to perform different operations on files (e.g. read, write, append, etc.). User – friendly : Python provides a user-friendly interface for file handling, making it easy to create, read and manipulate files. Cross-platform : Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.Disadvantages of File Handling in Python Error-prone: File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.). Security risks : File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system. Complexity : File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely. Performance : File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

In [ ]:
```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
print("my_list =", my_list)

# get a list with items from index 2 to index 4 (index 5 is not included)
print("my_list[2: 5] =", my_list[2: 5])

# get a list with items from index 2 to index -3 (index -2 is not included)
print("my_list[2: -2] =", my_list[2: -2])

# get a list with items from index 0 to index 2 (index 3 is not included)
print("my_list[0: 3] =", my_list[0: 3])
```

In [ ]:
```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
print("my_list =", my_list)

# get a list with items from index 5 to last
print("my_list[5: ] =", my_list[5: ])

# get a list from the first item to index -5
print("my_list[: -4] =", my_list[: -4])

# omitting both start and end index
# get a list from start to end items
print("my_list[:] =", my_list[:])
```

In [ ]:
```
fruits = ['apple', 'banana', 'orange']
print('Original List:', fruits)

fruits.append('cherry')

print('Updated List:', fruits)
```

In [ ]:
```
fruits = ['apple', 'banana', 'orange']
print("Original List:", fruits)

fruits.insert(2, 'cherry')

print("Updated List:", fruits)
```

In [ ]:
```
Python List Methods
Python has many useful list methods that make it really easy to work with lists.

Method         Description
append()         Adds an item to the end of the list
extend()         Adds items of lists and other iterables to the end of the list
insert()         Inserts an item at the specified index
remove()         Removes the specified value from the list
pop()        Returns and removes item present at the given index
clear()      Removes all items from the list
index()      Returns the index of the first matched item
count()      Returns the count of the specified item in the list
sort()       Sorts the list in ascending/descending order
```

```
reverse()       Reverses the item of the list
copy()      Returns the shallow copy of the list
```

In [ ]:
```
# creating a dictionary
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
```

In [ ]:
```
#Access Dictionary Items
#We can access the value of a dictionary item by placing the key inside square brac
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
  "England": "London"
}
# access the value of keys
print(country_capitals["Germany"])      # Output: Berlin
print(country_capitals["England"])      # Output: London
```

In [ ]:
```
#Add Items to a Dictionary
#We can add an item to a dictionary by assigning a value to a new key. For example,
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
}

# add an item with "Italy" as key and "Rome" as its value
country_capitals["Italy"] = "Rome"

print(country_capitals)
```

In [ ]:
```
Iterate Through a Dictionary
A dictionary is an ordered collection of items (starting from Python 3.7), therefor
We can iterate through dictionary keys one by one using a for loop.
```

In [ ]:
```
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome"
}

# print dictionary keys one by one
for country in country_capitals:
    print(country)

print()

# print dictionary values one by one
for country in country_capitals:
```

```
    capital = country_capitals[country]
    print(capital)
```

In [ ]:
```
Python Dictionary Methods
Here are some of the commonly used dictionary methods.

Function         Description
pop()        Removes the item with the specified key.
update()         Adds or changes dictionary items.
clear()      Remove all the items from the dictionary.
keys()       Returns all the dictionary's keys.
values()         Returns all the dictionary's values.
get()        Returns the value of the specified key.
popitem()        Returns the last inserted key and value as a tuple.
copy()       Returns a copy of the dictionary.
```

In [ ]:
```
The list data type has some more methods. Here are all of the methods of list objec

list.append(x)
Add an item to the end of the list. Similar to a[len(a):] = [x].

list.extend(iterable)
Extend the list by appending all the items from the iterable. Similar to a[len(a):]

list.insert(i, x)
Insert an item at a given position. The first argument is the index of the element

list.remove(x)
Remove the first item from the list whose value is equal to x. It raises a ValueErr

list.pop([i])
Remove the item at the given position in the list, and return it. If no index is sp

list.clear()
Remove all items from the list. Similar to del a[:].

list.index(x[, start[, end]])
Return zero-based index in the list of the first item whose value is equal to x. Ra

The optional arguments start and end are interpreted as in the slice notation and a

list.count(x)
Return the number of times x appears in the list.

list.sort(*, key=None, reverse=False)
Sort the items of the list in place (the arguments can be used for sort customizati

list.reverse()
Reverse the elements of the list in place.

list.copy()
Return a shallow copy of the list. Similar to a[:].
```

In [ ]:
```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
fruits.count('apple')
```

In [ ]:
```
fruits.count('tangerine')
```

In [ ]:
```
fruits.index('banana')
```

In [ ]:
```
fruits.index('banana', 4)  # Find next banana starting at position 4
```

In [ ]:
```
fruits.reverse()
fruits
```

In [ ]:
```
fruits.append('grape')
fruits
```

In [ ]:
```
fruits.sort()
fruits
```

In [ ]:
```
Assignment- Take stack of 5 numbers and add numbers.
```

In [16]:
```
# Input: An integer number
num = int(input("enter value for fact"))

# Initialize the factorial variable to 1
factorial = 1

# Calculate the factorial using a for loop
for i in range(1, num + 1):
    factorial *= i

# Output: The factorial of the number
print(f"The factorial of {num} is {factorial}")
#In Python, the f (or F) prefix before a string literal denotes an f-string, which
```

```
The factorial of 5 is 120
```

In [6]:
```
#only if condition
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

In [7]:
```
#if-else condition
i = 20

 # Checking if i is greater than 0
if (i > 0):
    print("i is positive")
else:
    print("i is 0 or Negative")
```

```
i is positive
```

In [ ]:
```
#IF age is more than 30 years and experience is more than 10 years then candidate i
```

In [9]:
```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

In [3]:
```
# a list of three elements
ages = [19, 26, 29]
print(ages)
```

```
[19, 26, 29]
```

In [9]:
```
list1=[1,2,3,4,5,5,3,2,'A','A','B']
print(list1)
```

```
[1, 2, 3, 4, 5, 5, 3, 2, 'A', 'A', 'B']
```

In [6]:
```
#List Characteristics
#In Python, lists are:

#Ordered - They maintain the order of elements.
#Mutable - Items can be changed after creation.
#Allow duplicates - They can contain duplicate values.
```

In [ ]:
```
#Access List Elements
#Each element in a list is associated with a number, known as an index. The index o
languages = ['Python', 'Java', 'C++']

# access the first element
print('languages[0] =', languages[0])

# access the third element
print('languages[2] =', languages[2])
```

In [ ]:
```
#Negative Indexing
#In Python, a list can also have negative indices. The index of the last element is
languages = ['Python', 'Java', 'C++']

# access the last item
print('languages[-1] =', languages[-1])

# access the third last item
print('languages[-3] =', languages[-3])
```

In [ ]:
```
#Slicing of a List in Python
#If we need to access a portion of a list, we can use the slicing operator, :
```

In [2]:
```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
print("my_list =", my_list)

# get a list with items from index 2 to index 4 (index 5 is not included)
print("my_list[2: 5] =", my_list[2: 5])

# get a list with items from index 2 to index -3 (index -2 is not included)
print("my_list[2: -2] =", my_list[2: -2])

# get a list with items from index 0 to index 2 (index 3 is not included)
print("my_list[0: 3] =", my_list[0: 3])
```

```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
my_list[2: 5] = ['o', 'g', 'r']
my_list[2: -2] = ['o', 'g', 'r']
my_list[0: 3] = ['p', 'r', 'o']
```

In [ ]:
```
#Omitting Start and End Indices in Slicing - If you omit the start index, the slici
```

In [3]:
```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
print("my_list =", my_list)

# get a list with items from index 5 to last
```

```
print("my_list[5: ] =", my_list[5: ])

# get a list from the first item to index -5
print("my_list[: -4] =", my_list[: -4])

# omitting both start and end index
# get a list from start to end items
print("my_list[:] =", my_list[:])
```

```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
my_list[5: ] = ['a', 'm']
my_list[: -4] = ['p', 'r', 'o']
my_list[:] = ['p', 'r', 'o', 'g', 'r', 'a', 'm']
```

In [ ]:
```
#Add Elements to a Python List - As mentioned earlier, lists are mutable and we can
```

In [6]:
```
fruits = ['apple', 'banana', 'orange']
print('Original List:', fruits)

fruits.append('cherry')

print('Updated List:', fruits)
```

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'orange', 'cherry']
```

In [ ]:
```
#Add Elements at the Specified Index We can insert an element at the specified inde
```

In [5]:
```
fruits = ['apple', 'banana', 'orange']
print("Original List:", fruits)

fruits.insert(2, 'cherry')

print("Updated List:", fruits)
```

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'cherry', 'orange']
```

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

Method Description append() Adds an item to the end of the list extend() Adds items of lists and other iterables to the end of the list insert() Inserts an item at the specified index remove() Removes the specified value from the list pop() Returns and removes item present at the given index clear() Removes all items from the list index() Returns the index of the first matched item count() Returns the count of the specified item in the list sort() Sorts the list in ascending/descending order reverse() Reverses the item of the list copy() Returns the shallow copy of the list

Python Dictionary A Python dictionary is a collection of items, similar to lists and tuples. However, unlike lists and tuples, each item in a dictionary is a key-value pair (consisting of a key and a value).

In [7]:
```
# creating a dictionary
country_capitals = {
  "Germany": "Berlin",
```

```
  "Canada": "Ottawa",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
```

```
{'Germany': 'Berlin', 'Canada': 'Ottawa', 'England': 'London'}
```

In [8]:
```
#Access Dictionary Items
#We can access the value of a dictionary item by placing the key inside square brac
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
  "England": "London"
}
# access the value of keys
print(country_capitals["Germany"])      # Output: Berlin
print(country_capitals["England"])      # Output: London
```

```
Berlin
London
```

In [ ]:
```
#Add Items to a Dictionary
#We can add an item to a dictionary by assigning a value to a new key. For example,
country_capitals = {
  "Germany": "Berlin",
  "Canada": "Ottawa",
}

# add an item with "Italy" as key and "Rome" as its value
country_capitals["Italy"] = "Rome"

print(country_capitals)
```

Iterate Through a Dictionary A dictionary is an ordered collection of items (starting from Python 3.7), therefore it maintains the order of its items. We can iterate through dictionary keys one by one using a for loop.

In [1]:
```
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome"
}

# print dictionary keys one by one
for country in country_capitals:
    print(country)

print()

# print dictionary values one by one
for country in country_capitals:
    capital = country_capitals[country]
    print(capital)
```

```
United States
Italy

Washington D.C.
Rome
```

Python Dictionary Methods Here are some of the commonly used dictionary methods.

Function Description pop() Removes the item with the specified key. update() Adds or changes dictionary items. clear() Remove all the items from the dictionary. keys() Returns all the dictionary's keys. values() Returns all the dictionary's values. get() Returns the value of the specified key. popitem() Returns the last inserted key and value as a tuple. copy() Returns a copy of the dictionary.