UNIT-I

**Python basics, Objects**- Python Objects, Standard Types, Other Built-in Types, Internal Types, Standard Type Operators, Standard Type Built-in Functions, Categorizing the Standard Types, Unsupported Types

**Numbers** - Introduction to Numbers, Integers, Floating Point Real Numbers, Complex Numbers, Operators, Built-in Functions, Related Modules

**Sequences** - Strings, Lists, and Tuples, Dictionaries and Set Types Control Flow, Truthiness, Sorting, List Comprehensions, Generators and Iterators

UNIT-II

**Files:** File Objects, File Built-in Function [ open() ], File Built-in Methods, File Built-in Attributes,Standard Files, Command-line Arguments, File System, File Execution

**Exceptions:** Exceptions in Python, Detecting and Handling Exceptions, Context Management,

Exceptions as Strings, Raising Exceptions, Assertions, Standard Exceptions, Creating Exceptions,Why Exceptions (Now)?, Why Exceptions at All?, Exceptions and the sys Module, Related modules

**Modules:** Modules and Files, Namespaces, Importing Modules, Importing Module Attributes, Module Built-in Functions, Packages, Other Features of Modules

UNIT-III

**Regular Expressions:** Introduction, Special Symbols and Characters, Res and pythonMultithreaded

**Programming:** Introduction, Threads and Processes, Python, Threads, and the Global Interpreter Lock, Thread Module, Threading Module, Related Modules

UNIT-IV

**GUI Programming**: Introduction, Tkinter and Python Programming, Brief Tour of Other GUIs,

Related Modules and Other GUIs

**Web Programming**: Introduction, Wed Surfing with Python, Creating Simple Web Clients,

Advanced Web Clients, CGI-Helping Servers Process Client Data, Building CGI Application,

Advanced CGI, Web (HTTP) Servers

UNIT-V

**Database Programming**: Introduction, Python Database Application Programmer’s Interface

(DBAPI), Object Relational Managers (ORMs), Related Modules

UNIT – 1

* Introduction to python

Python is a widely used general-purpose, high-level programming language. It was initially designed by **Guido van Rossum**in **1991**and developed by Python Software Foundation. It was mainly developed to emphasize code readability, and its syntax allows programmers to express concepts in fewer lines of code.

**History**

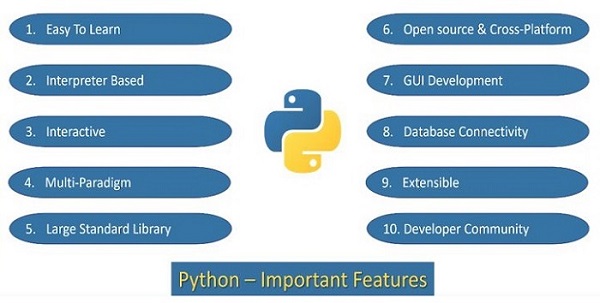
In the late 1980s, history was about to be written. It was that time when working on Python started. Soon after that, Guido Van Rossum began doing its application-based work in December of 1989 at Centrum Wiskunde & Informatica (CWI) which is situated in the Netherlands.

The inspiration for the name came from the BBC’s TV Show – ‘ **Monty Python’s Flying Circus’**, as he was a big fan of the TV show and also he wanted a short, unique and slightly mysterious name for his invention and hence he named it Python! He was the “Benevolent dictator for life” (BDFL) until he stepped down from the position as the leader on 12th July 2018. For quite some time he used to work for Google, but currently, he is working at Dropbox.

The language was finally released in 1991. When it was released, it used a lot fewer codes to express the concepts, when we compare it with [Java](https://www.geeksforgeeks.org/java), [C++](https://www.geeksforgeeks.org/c-plus-plus)& [C](https://www.geeksforgeeks.org/c-programming-language). Its design philosophy was quite good too. Its main objective is to provide code readability and advanced developer productivity. When it was released, it had more than enough capability to provide classes with inheritance, several core data types of exception handling and functions.

* **Features of python**

Python is a feature-rich, high-level, interpreted, interactive, and object-oriented scripting language. Python is a versatile and very popular programming language due to its features such as readability, simplicity, extensive libraries, and many more.



Python's most important features are as follows:

* [Easy to Learn](https://www.tutorialspoint.com/python/python_features.htm#easy-to-learn)
* [Dynamically Typed](https://www.tutorialspoint.com/python/python_features.htm#dynamically-typed)
* [Interpreter Based](https://www.tutorialspoint.com/python/python_features.htm#interpreter-based)
* [Interactive](https://www.tutorialspoint.com/python/python_features.htm#interactive)
* [Multi-paradigm](https://www.tutorialspoint.com/python/python_features.htm#multi-paradigm)
* [Standard Library](https://www.tutorialspoint.com/python/python_features.htm#standard-library)
* [Open Source and Cross Platform](https://www.tutorialspoint.com/python/python_features.htm#open-source-and-cross-platform)
* [GUI Applications](https://www.tutorialspoint.com/python/python_features.htm#gui-applications)
* [Database Connectivity](https://www.tutorialspoint.com/python/python_features.htm#database-connectivity)
* [Extensible](https://www.tutorialspoint.com/python/python_features.htm#extensible)

1. Easy to Learn

This is one of the most important reasons for the popularity of Python. Python has a limited set of keywords. Its features such as simple [syntax](https://www.tutorialspoint.com/python/python_basic_syntax.htm), usage of indentation to avoid clutter of curly brackets and dynamic typing that doesn't necessitate prior declaration of variable help a beginner to learn Python quickly and easily.

1. Dynamically Typed

Python is a dynamically typed programming language. In Python, you don't need to specify the variable time at the time of the variable declaration. The types are specified at the runtime based on the assigned value due to its dynamically typed feature.

1. Interpreter Based

Instructions in any programming languages must be translated into machine code for the processor to execute them. Programming languages are either compiler based or interpreter based.

In case of a compiler, a [machine language](https://www.tutorialspoint.com/machine_learning/index.htm) version of the entire source program is generated. The conversion fails even if there is a single erroneous statement. Hence, the development process is tedious for the beginners. The C family languages (including [C](https://www.tutorialspoint.com/cprogramming/index.htm), [C++](https://www.tutorialspoint.com/cplusplus/index.htm), [Java](https://www.tutorialspoint.com/java/index.htm), [C#](https://www.tutorialspoint.com/csharp/index.htm) etc) are compiler based.

Python is an interpreter based language. The interpreter takes one instruction from the source code at a time, translates it into machine code and executes it. Instructions before the first occurrence of error are executed. With this feature, it is easier to debug the program and thus proves useful for the beginner level programmer to gain confidence gradually. Python therefore is a beginner-friendly language.

1. Interactive

Standard Python distribution comes with an interactive shell that works on the principle of REPL (Read Evaluate Print Loop). The shell presents a Python prompt >>>. You can type any valid Python expression and press Enter. Python interpreter immediately returns the response and the prompt comes back to read the next expression.

>>> 2\*3+1

7

>>> print ("Hello World")

Hello World

The interactive mode is especially useful to get familiar with a library and test out its functionality. You can try out small code snippets in interactive mode before writing a program.

1. Multi-paradigm

Python is a completely [object-oriented](https://www.tutorialspoint.com/python/python_oops_concepts.htm) language. Everything in a Python program is an [object](https://www.tutorialspoint.com/python/python_object_classes.htm). However, Python conveniently encapsulates its object orientation to be used as an imperative or procedural language such as C. Python also provides certain functionality that resembles functional programming. Moreover, certain third-party tools have been developed to support other programming paradigms such as aspect-oriented and logic programming.

1. Standard Library

Even though it has a very few keywords (only Thirty Five), Python software is distributed with a standard library made of large number of modules and packages. Thus Python has out of box support for programming needs such as serialization, data compression, internet data handling, and many more. Python is known for its batteries included approach.

Some of the Python's popular modules are:

* [NumPy](https://www.tutorialspoint.com/numpy/index.htm)
* [Pandas](https://www.tutorialspoint.com/python_pandas/index.htm)
* [Matplotlib](https://www.tutorialspoint.com/matplotlib/index.htm)
* Tkinter
* [Math](https://www.tutorialspoint.com/python/python_maths.htm)

1. Open Source and Cross Platform

Python's standard distribution can be downloaded from <https://www.python.org/downloads/> without any restrictions. You can download pre-compiled binaries for various operating system platforms. In addition, the source code is also freely available, which is why it comes under open source category.

Python is a cross-platform language. Pre-compiled binaries are available for use on various operating system platforms such as [Windows](https://www.tutorialspoint.com/windows10/index.htm), [Linux](https://www.tutorialspoint.com/unix/index.htm), Mac OS, [Android OS](https://www.tutorialspoint.com/android/index.htm). The reference implementation of Python is called CPython and is written in C. You can download the source code and compile it for your OS platform.

A Python program is first compiled to an intermediate platform independent byte code. The virtual machine inside the interpreter then executes the byte code. This behaviour makes Python a cross-platform language, and thus a Python program can be easily ported from one OS platform to other.

1. GUI Applications

Python's standard distribution has an excellent graphics library called TKinter. It is a Python port for the vastly popular GUI toolkit called TCL/Tk. You can build attractive user-friendly GUI applications in Python. GUI toolkits are generally written in C/C++. Many of them have been ported to Python. Examples are [PyQt](https://www.tutorialspoint.com/pyqt/index.htm" \t "_blank), [WxWidgets](https://www.tutorialspoint.com/wxpython/index.htm" \t "_blank), [PySimpleGUI](https://www.tutorialspoint.com/pysimplegui/index.htm" \t "_blank) etc.

1. Database Connectivity

Almost any type of database can be used as a backend with the Python application. DB-API is a set of specifications for database driver software to let Python communicate with a relational database. With many third party libraries, Python can also work with NoSQL databases such as [MongoDB](https://www.tutorialspoint.com/mongodb/index.htm).

1. Extensible

The term extensibility implies the ability to add new features or modify existing features. As stated earlier, CPython (which is Python's reference implementation) is written in C. Hence one can easily write modules/libraries in C and incorporate them in the standard library. There are other implementations of Python such as Jython (written in Java) and [IPython](https://www.tutorialspoint.com/jupyter/ipython_introduction.htm" \t "_blank) (written in C#). Hence, it is possible to write and merge new functionality in these implementations with Java and C# respectively.

* Applications of Python

1. Operating Systems

The robust standard library of Python makes it perfect for building entire operating systems. The object-oriented design of the language ensures large projects are easily managed. Python is compatible with most operating systems and can be easily used to build native applications for Windows and Mac computers.

Example

While Python is not typically used to build entire operating systems, it is often used for scripting and automation tasks within operating systems. Python scripts can automate system administration tasks, manage files and directories, and interact with system APIs. Examples include writing scripts to automate backups, manage user accounts, or monitor system performance.

2. Web Development

Python offers several choices for complex web development projects. HTML and JavaScript are the main languages used to build the front end of an application. But Python-based web frameworks like Django, Pyramid, and Flask are used to handle backend or server-side functionality of sites and services like Spotify, Reddit, and Mozilla. Giant platforms like Google and YouTube depend largely on Python for critical infrastructure.

The standard library of Python also supports many Internet protocols like HTML and XML, JSON, Email processing, FTP, and IMAP.

Example

Python is widely used in web development for building dynamic websites, web applications, and APIs. Frameworks like Django and Flask provide powerful tools for building web applications, handling HTTP requests, and interacting with databases. Popular websites and web applications like Instagram, Pinterest, and Spotify are built using Python and its web frameworks.

3. Game Development

Just like for web development, Python offers an array of tools and libraries for game development. Would you believe, Battlefield 2 – one of the most popular shooting games of the early 2000s, was developed with the use of Python.

Python’s 2D and 3D game development libraries are PyGame, Pycap, Panda#D, Construct, PySoy, and PyOpenGL.

Python has been used to develop popular games, including Sims 4, World of Tanks, Eve Online, Mount & Blade, Doki Doki Literature Club, and Disney’s Toontown Online, to name a few.

Example

Python is increasingly used in the game development industry for building games, game engines, and game development tools. Libraries like Pygame provide a framework for building 2D games, while engines like Panda3D and Godot support the development of both 2D and 3D games. Python's simplicity and ease of use make it an attractive choice for prototyping and rapid game development.

4. Scientific and Numeric Computing

The Python ecosystem offers numerous tools and libraries that help scientists and researchers in scientific and numeric computing.

* SciPy is a set of packages for mathematics, science, and engineering
* Pandas is a library used for data analysis and modeling
* IPython is a strong interactive shell that provides hassle-free editing and recording of a work session and aid in visualizing and parallel computing.
* FreeCAD and Abaqus are real-life numerical and scientific applications built with Python

Example

Python is widely used in scientific computing and data analysis due to its rich ecosystem of libraries and tools. Libraries like NumPy, SciPy, and Pandas provide powerful tools for numerical computing, data manipulation, and statistical analysis. Python is also used in scientific research, engineering simulations, and data visualization tasks.

5. Artificial Intelligence and Machine Learning

The hottest buzzwords of the decade – Artificial Intelligence (AI) and Machine Learning are mostly about algorithms, code, and logic. Python, along with a few other programming languages, is increasingly being used for developing AI and ML-powered solutions. The scope and power of Python, along with its stability and security, make it ideal for running AI and ML systems.

Some important libraries for the job are:

* Scikit-Learn – for building various machine learning models
* SciPy – for scientific and technical computing
* TensorFlow – for state-of-the-art neural networks
* Keras – for artificial neural networks
* Pandas – for data analysis and manipulation

Example

Python is the preferred language for artificial intelligence (AI) and machine learning (ML) applications due to its simplicity, flexibility, and extensive libraries. Frameworks like TensorFlow, PyTorch, and scikit-learn provide tools for building and training machine learning models, neural networks, and deep learning algorithms. Python's popularity in AI and ML has led to its widespread adoption in areas like natural language processing, computer vision, and predictive analytics.

6. Desktop GUI

Python is an excellent choice for desktop GUI (Graphical User Interface) programming. The language offers numerous options for developers to build a fully functional GUI. The comprehensive syntax and modular programming approach of the Python framework help create a super-fast and responsive GUI.

Some prominent applications of Python tools for GUI development are PyQt, Tkinter, wxWidgets, Python GTK+, and Kivy. Standard applications like Dropbox and BitTorrent are primarily written in Python.

Example

Python can be used to develop desktop graphical user interface (GUI) applications using libraries like Tkinter, PyQt, and wxPython. These libraries provide tools for creating windows, dialogs, buttons, and other GUI components. Python's simplicity and cross-platform compatibility make it suitable for building desktop applications for various operating systems.

7. Business Applications

Applications of Python also include building ERP and ecommerce systems. Business applications are different from typical consumer software because they offer a set of specific features instead of a variety of features. Besides, they target a very tight-knit user group, usually an organization.

Python is perfect for delivering best-performance custom solutions for business applications as well as consumer applications.

Odoo is a well-rounded management software that offers numerous business applications that constitute a complete set of enterprise management applications.

Tryton is a three-tier high-level application platform designed for general purposes.

Example

Python is used in business applications for a wide range of purposes, including data analysis, automation, and web development. Python scripts can automate repetitive tasks, generate reports, and interact with business databases and APIs. Python's versatility and ease of integration with other technologies make it a valuable tool for building custom business applications.

8. Console-based Application

Console-based applications are text-based programs that interact with users through a command-line interface. These applications are typically used for tasks such as data processing, system administration, or automation scripts.

Example

A simple console-based calculator program written in Python. This program prompts the user to enter mathematical expressions and evaluates them to produce the result.

# Simple console-based calculator

while True:

 expression = input("Enter a mathematical expression (or 'quit' to exit): ")

 if expression.lower() == 'quit':

    break

try:

    result = eval(expression)

    print("Result:", result)

except:

    print("Invalid expression. Please try again.")

9. Audio or Video-based Applications

Python can be used to create applications that manipulate audio or video data, such as media players, editors, or streaming services. Libraries like PyDub and MoviePy provide tools for processing audio and video files in Python.

Example

A simple audio player application written in Python using the PyDub library. This application allows users to play, pause, and stop audio files.

from pydub import AudioSegment

from pydub.playback import play

# Load audio file

audio\_file = AudioSegment.from\_file("C:/mani/example.mp3")

# Play audio

play(audio\_file)

10. 3D CAD Applications

Python can be used in 3D computer-aided design (CAD) applications for tasks such as modeling, rendering, and simulation. Libraries like Blender provide a comprehensive set of tools for creating and manipulating 3D models programmatically.

Example

A script written in Python to create a simple 3D model using Blender's scripting interface. This script generates a cube and exports it to a file in the Wavefront OBJ format.

import bpy

# Create a cube

bpy.ops.mesh.primitive\_cube\_add(size=2)

# Export the cube to OBJ format

bpy.ops.export\_scene.obj(filepath="cube.obj")

11. Image Processing

Python is commonly used for image processing tasks such as image enhancement, segmentation, and object detection. Libraries like OpenCV and Pillow provide tools for manipulating and analyzing images in Python.

Example

A simple image processing application written in Python using the OpenCV library. This application loads an image, converts it to grayscale, and applies a Gaussian blur filter to smooth the image.

import cv2

# Load image

image = cv2.imread("example.jpg")

# Convert image to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2YCrCb)

# Apply Gaussian blur

blurred\_image = cv2.GaussianBlur(gray\_image, (5, 5), 0)

# Display original and processed images

cv2.imshow("Original Image", image)

cv2.imshow("Blurred Image", blurred\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

* Python objects

In Python, **identity**, **type**, and **value** are three key characteristics that every object has. Understanding these properties is fundamental to how Python works under the hood.

Let's break them down:

**1. Identity**

* The **identity** of an object refers to its unique identity in memory. It is an identifier that distinguishes one object from another. Every object created in Python is stored at a specific location in memory, and its identity corresponds to that memory location.
* You can access an object's identity using the id() function. The value returned by id() is the memory address where the object is stored, but it is represented as an integer.
* The identity of an object is **constant** during the lifetime of the object.

**Example**:

a = "hello"

b = a

print(id(a))

print(id(b))

output:

1522483510448

1522483510448

* In this example, both a and b refer to the same string "hello", so their identities (memory addresses) are the same.

**2. Type**

* The **type** of an object refers to the class or data type the object belongs to. The type defines what operations can be performed on the object and how it behaves. Every object in Python has a type that can be accessed using the type() function.
* The type helps determine the structure and behavior of the object, such as whether it's an integer, string, list, or a custom class instance.

**Example**:

a = 10

b = "hello"

print(type(a))

print(type(b))

output:

<class 'int'>

<class 'str'>

* In this example, a is of type int, and b is of type str.

**3. Value**

* The **value** of an object refers to the actual data that the object holds. The value is the content of the object, such as a number, string, list, etc.
* The value of an object can change if the object is mutable (like a list), but for immutable objects (like strings and tuples), the value cannot be changed after creation.

**Example**:

a = 10

b = 20

print(a)

print(b)

a = 30

print(a)

output:

10

20

30

## Python Data Types/ Standard Types

**Python data types** are actually classes, and the defined variables are their instances or objects. Since Python is dynamically typed, the data type of a variable is determined at runtime based on the assigned value.

In general, the data types are used to define the type of a variable. It represents the type of data we are going to store in a variable and determines what operations can be done on it.

Each programming language has its own classification of data items. With these datatypes, we can store different types of data values.

## Python Numeric Data Types

Python numeric data types store numeric values. Number objects are created when you assign a value to them. For example −

a = 10      # int data type

b = 10.023  # float data type

c = 10+3j   # complex data type

print(a)

print(b)

print(c)

output:

10

10.023

(10+3j)

Python supports three different numerical types and each of them have built-in classes in Python library, called int, float and complex respectively −

* int (signed integers)
* float (floating point real values)
* complex (complex numbers)

Here are some examples of numbers −

|  |  |  |
| --- | --- | --- |
| **int** | **float** | **complex** |
| 10 | 0.0 | 3.14j |
| 0O777 | 15.20 | 45.j |

## Python Sequence Data Types

Sequence is a collection data type. It is an ordered collection of items. Items in the sequence have a positional index starting with 0. It is conceptually similar to an array in C or C++. There are following three sequence data types defined in Python.

* List Data Type
* Tuple Data Type
* Range Data Type

Python sequences are bounded and iterable - Whenever we say an iterable in Python, it means a sequence data type (for example, a list).

### (a) Python List Data Type

[Python Lists](https://www.tutorialspoint.com/python/python_lists.htm) are the most versatile compound data types. A Python list contains items separated by commas and enclosed within square brackets ([]). To some extent, Python lists are similar to arrays in C. One difference between them is that all the items belonging to a Python list can be of different data type where as C array can store elements related to a particular data type.

languages = ["Swift", "Java", "Python"]

print(languages)

output:

['Swift', 'Java', 'Python']

### (b) Python Tuple Data Type

[Python tuple](https://www.tutorialspoint.com/python/python_tuples.htm) is another sequence data type that is similar to a list. A Python tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses (...).

A tuple is also a sequence, hence each item in the tuple has an index referring to its position in the collection. The index starts from 0.

product = ('Xbox', 499.99)

print(product)

output:

('Xbox', 499.99)

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed i.e. lists are mutable, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated (immutable). Tuples can be thought of as **read-only** lists.

### (c) Python Range Data Type

A Python range is an immutable sequence of numbers which is typically used to iterate through a specific number of items.

It is represented by the **Range** class. The constructor of this class accepts a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number. Following is the syntax of the function −

range(start, stop, step)

Here is the description of the parameters used −

* **start**: Integer number to specify starting position,(Its optional, Default: 0)
* **stop**: Integer number to specify ending position (It's mandatory)
* **step**: Integer number to specify increment, (Its optional, Default: 1)

### Example of Range Data Type

Following is a program which uses for loop to print number from 0 to 4 –

for i in range(5):

  print(i)

output:

0

1

2

3

4

Again, let's modify the program to print the number starting from 1 but with an increment of 2 instead of 1:

for i in range(1, 10, 2):

  print(i)

output:

1

3

5

7

9

1. Python String Data Type

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

name = 'Python'

print(name)

message = 'Python programmers'

print(message)

output:

Python

Python programmers

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

## Python Set Data Type

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

student\_id = {112, 114, 116, 118, 115}

print(student\_id)

print(type(student\_id))

output:

{112, 114, 115, 116, 118}

<class 'set'>

Here, we have created a set named **student\_id** 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,

capital\_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital\_city)

output:

{'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

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'

## Python Boolean Data Type

Python **boolean** type is one of built-in data types which represents one of the two values either **True** or **False**. Python **bool()** function allows you to evaluate the value of any expression and returns either True or False based on the expression.

A Boolean number has only two possible values, as represented by the keywords, **True** and **False**. They correspond to integer 1 and 0 respectively.

### **Example of Boolean Data Type**

Following is a program which prints the value of boolean variables a and b −

a = True

print(a)

print(type(a))

output:

true

<class 'bool'>

* **Other built-in types**

Besides the **core built-in types** (like int, float, str, list, dict, etc.), Python includes **other built-in types** that are more advanced or used in specific contexts. These are still part of the Python language and are available without needing to import additional libraries.

1. Binary Types in Python

Binary types are used to store and manipulate **binary data** (such as files, images, or raw network streams). Python provides three main binary types:

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Mutable | Description | Example |
| bytes | No | Immutable sequence of bytes | b'hello' |
| bytearray | Yes | Mutable sequence of bytes | bytearray(b'hello') |
| memoryview | Depends | View into another binary object’s memory | memoryview(b'abc') |

## bytes

Bytes are an **immutable** sequence of bytes,Each byte is a number between 0 and 255.

Commonly used when reading binary files, working with byte-oriented protocols, etc.

* Similar to a string, but contains raw 8-bit values.
* Represented with a prefix b or B before the string.

### Example:

b = b'hello'

print(b[0])

print(type(b))

output:

104

<class 'bytes'>

## bytearray

Bytearray A **mutable** sequence of bytes and Allows modification of byte content.

Useful when you need to change binary data.

* Can be created from bytes, lists of integers, or a string (with encoding).

### Example:

ba = bytearray(b'hello')

ba[0] = 72

print(ba)

output:

bytearray(b'Hello')

## memoryview

memoryview Provides a memory-efficient **view** into the data of a binary object like bytes or bytearray and Allows access to parts of the data **without copying**. These Useful for large datasets and performance-critical applications.

* Acts like a "window" on the data.
* Can read and write (if the original object is mutable).

### Example:

data = bytearray(b'hello')

mv = memoryview(data)

print(mv)

mv[0] = 72

print(data)

output:

<memory at 0x000001EB0C437940>

bytearray(b'Hello')

1. Special Types in Python

Python provides some **special types** that are not commonly used in basic programming but are **essential in advanced applications**, such as operator overloading, placeholder code, class design, or memory optimization.

## NoneType

The type of the None object. It is Represents the **absence of a value** or a **null value**.

Often used as a **default return value** of functions that don’t return anything.

### Example:

print(type(None))

output:

<class 'NoneType'>

## type

A special type that represents the **type of all objects** and Can also be used to **dynamically create classes**.

### Example:

print(type(5))

output:

<class 'int'>

## object

The **base class of all classes** in Python. And All user-defined and built-in classes derive from object.

class A:

    pass

print(isinstance(A(), object))

output:

True

## slice

Represents a **slice object**, used in list/string slicing operations. And Can be created manually using slice(start, stop, step).

### Example:

s = slice(1, 5, 2)

print([0, 1, 2, 3, 4, 5][s])

output:

[1, 3]

* Python Operators

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

**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.

|  |  |  |
| --- | --- | --- |
| Operator | Operation | Example |
| + | Addition | 5 + 2 = 7 |
| - | Subtraction | 4 - 2 = 2 |
| \* | Multiplication | 2 \* 3 = 6 |
| / | Division | 4 / 2 = 2 |
| // | Floor Division | 10 // 3 = 3 |
| % | Modulo | 5 % 2 = 1 |
| \*\* | Power | 4 \*\* 2 = 16 |

**Example 1: Arithmetic Operators in Python**

a = 7

b = 2

# addition

print ('Sum: ', a + b)

# subtraction

print ('Subtraction: ', a - b)

# multiplication

print ('Multiplication: ', a \* b)

# division

print ('Division: ', a / b)

# floor division

print ('Floor Division: ', a // b)

# modulo

print ('Modulo: ', a % b)

# a to the power b

print ('Power: ', a \*\* b)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Sum: 9

Subtraction: 5

Multiplication: 14

Division: 3.5

Floor Division: 3

Modulo: 1

Power: 49

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,

# assign 5 to x

x = 5

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

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

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| = | Assignment Operator | a = 7 |
| += | Addition Assignment | a += 1 # a = a + 1 |
| -= | Subtraction Assignment | a -= 3 # a = a - 3 |
| \*= | Multiplication Assignment | a \*= 4 # a = a \* 4 |
| /= | Division Assignment | a /= 3 # a = a / 3 |
| %= | Remainder Assignment | a %= 10 # a = a % 10 |
| \*\*= | Exponent Assignment | a \*\*= 10 # a = a \*\* 10 |

**Example 2: Assignment Operators**

a = 10

b = 5

a += b

print(a)

**Output**

15

[Run Code](https://www.programiz.com/python-programming/online-compiler)

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

Similarly, we can use any other assignment operators as per our needs.

**3. Python Comparison Operators**

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

a = 5

b = 2

print (a > b) # True

[Run Code](https://www.programiz.com/python-programming/online-compiler)

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

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| == | Is Equal To | 3 == 5 gives us False |
| != | Not Equal To | 3 != 5 gives us True |
| > | Greater Than | 3 > 5 gives us False |
| < | Less Than | 3 < 5 gives us True |
| >= | Greater Than or Equal To | 3 >= 5 give us False |
| <= | Less Than or Equal To | 3 <= 5 gives us True |

**Example 3: Comparison Operators**

a = 5

b = 2

print('a == b =', a == b)

print('a != b =', a != b)

print('a > b =', a > b)

print('a < b =', a < b)

print('a >= b =', a >= b)

print('a <= b =', a <= b)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

a == b = False

a != b = True

a > b = True

a < b = False

a >= b = True

a <= b = False

**Note:** Comparison operators are used in decision-making and [loops](https://www.programiz.com/python-programming/looping-technique). 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,

a = 5

b = 6

print((a > 2) and (b >= 6))

**Output**

True

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

|  |  |  |
| --- | --- | --- |
| Operator | Example | Meaning |
| and | a **and** b | **Logical AND**: True only if both the operands are True |
| or | a **or** b | **Logical OR**: True if at least one of the operands is True |
| not | **not** a | **Logical NOT**: True if the operand is False and vice-versa. |

**Example 4: Logical Operators**

n = 5

print(n > 3 and n < 10)

print(n > 3 or n < 4)

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

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

True

True

False

**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)

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| & | Bitwise AND | x & y = 0 (0000 0000) |
| | | Bitwise OR | x | y = 14 (0000 1110) |
| ~ | Bitwise NOT | ~x = -11 (1111 0101) |
| ^ | Bitwise XOR | x ^ y = 14 (0000 1110) |
| >> | Bitwise right shift | x >> 2 = 2 (0000 0010) |
| << | Bitwise left shift | x 0010 1000) |

**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 are used to check if two values are located at the same memory location.

It's important to note that having two variables with equal values doesn't necessarily mean they are identical.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| is | True if the operands are identical (refer to the same object) | x is True |
| is not | True if the operands are not identical (do not refer to the same object) | x is not True |

**Example 4: Identity operators in Python**

x1 = 5

y1 = 5

x2 = 'Hello'

y2 = 'Hello'

x3 = [1,2,3]

y3 = [1,2,3]

print(x1 is not y1)

print(x2 is y2)

print(x3 is y3)

**Output**

False

True

False

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Here, we see that x1 and y1 are integers of the same values, so they are equal as well as identical. The same is the case with x2 and y2 (strings).

But x3 and y3 are lists. They are equal but not identical. It is because the interpreter locates them separately in memory, although they are equal.

**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](https://www.programiz.com/python-programming/string), [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [set](https://www.programiz.com/python-programming/set) and [dictionary](https://www.programiz.com/python-programming/dictionary)).

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

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| in | True if value/variable is **found** in the sequence | 5 in x |
| not in | True if value/variable is **not found** in the sequence | 5 not in x |

**Example 5: Membership operators in Python**

message = 'Hello world'

dict1 = {1:'a', 2:'b'}

print('H' in message)

print('hello' not in message)

print(1 in dict1)

print('a' in dict1)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

True

True

True

False

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

Similarly, 1 is key, and 'a' is the value in dictionary dict1. Hence, 'a' in y returns False.

* Python Built in Functions

Python has a set of built-in functions.

|  |  |
| --- | --- |
| **Function** | **Description** |
| [abs()](https://www.w3schools.com/python/ref_func_abs.asp) | Returns the absolute value of a number |
| [all()](https://www.w3schools.com/python/ref_func_all.asp) | Returns True if all items in an iterable object are true |
| [any()](https://www.w3schools.com/python/ref_func_any.asp) | Returns True if any item in an iterable object is true |
| [ascii()](https://www.w3schools.com/python/ref_func_ascii.asp) | Returns a readable version of an object. Replaces none-ascii characters with escape character |
| [bin()](https://www.w3schools.com/python/ref_func_bin.asp) | Returns the binary version of a number |
| [bool()](https://www.w3schools.com/python/ref_func_bool.asp) | Returns the boolean value of the specified object |
| [bytearray()](https://www.w3schools.com/python/ref_func_bytearray.asp) | Returns an array of bytes |
| [bytes()](https://www.w3schools.com/python/ref_func_bytes.asp) | Returns a bytes object |
| [callable()](https://www.w3schools.com/python/ref_func_callable.asp) | Returns True if the specified object is callable, otherwise False |
| [chr()](https://www.w3schools.com/python/ref_func_chr.asp) | Returns a character from the specified Unicode code. |
| classmethod() | Converts a method into a class method |
| [compile()](https://www.w3schools.com/python/ref_func_compile.asp) | Returns the specified source as an object, ready to be executed |
| [complex()](https://www.w3schools.com/python/ref_func_complex.asp) | Returns a complex number |
| [delattr()](https://www.w3schools.com/python/ref_func_delattr.asp) | Deletes the specified attribute (property or method) from the specified object |
| [dict()](https://www.w3schools.com/python/ref_func_dict.asp) | Returns a dictionary (Array) |
| [dir()](https://www.w3schools.com/python/ref_func_dir.asp) | Returns a list of the specified object's properties and methods |
| [divmod()](https://www.w3schools.com/python/ref_func_divmod.asp) | Returns the quotient and the remainder when argument1 is divided by argument2 |
| [enumerate()](https://www.w3schools.com/python/ref_func_enumerate.asp) | Takes a collection (e.g. a tuple) and returns it as an enumerate object |
| [eval()](https://www.w3schools.com/python/ref_func_eval.asp) | Evaluates and executes an expression |
| [exec()](https://www.w3schools.com/python/ref_func_exec.asp) | Executes the specified code (or object) |
| [filter()](https://www.w3schools.com/python/ref_func_filter.asp) | Use a filter function to exclude items in an iterable object |
| [float()](https://www.w3schools.com/python/ref_func_float.asp) | Returns a floating point number |
| [format()](https://www.w3schools.com/python/ref_func_format.asp) | Formats a specified value |
| [frozenset()](https://www.w3schools.com/python/ref_func_frozenset.asp) | Returns a frozenset object |
| [getattr()](https://www.w3schools.com/python/ref_func_getattr.asp) | Returns the value of the specified attribute (property or method) |
| [globals()](https://www.w3schools.com/python/ref_func_globals.asp) | Returns the current global symbol table as a dictionary |
| [hasattr()](https://www.w3schools.com/python/ref_func_hasattr.asp) | Returns True if the specified object has the specified attribute (property/method) |
| hash() | Returns the hash value of a specified object |
| help() | Executes the built-in help system |
| [hex()](https://www.w3schools.com/python/ref_func_hex.asp) | Converts a number into a hexadecimal value |
| [id()](https://www.w3schools.com/python/ref_func_id.asp) | Returns the id of an object |
| [input()](https://www.w3schools.com/python/ref_func_input.asp) | Allowing user input |
| [int()](https://www.w3schools.com/python/ref_func_int.asp) | Returns an integer number |
| [isinstance()](https://www.w3schools.com/python/ref_func_isinstance.asp) | Returns True if a specified object is an instance of a specified object |
| [issubclass()](https://www.w3schools.com/python/ref_func_issubclass.asp) | Returns True if a specified class is a subclass of a specified object |
| [iter()](https://www.w3schools.com/python/ref_func_iter.asp) | Returns an iterator object |
| [len()](https://www.w3schools.com/python/ref_func_len.asp) | Returns the length of an object |
| [list()](https://www.w3schools.com/python/ref_func_list.asp) | Returns a list |
| [locals()](https://www.w3schools.com/python/ref_func_locals.asp) | Returns an updated dictionary of the current local symbol table |
| [map()](https://www.w3schools.com/python/ref_func_map.asp) | Returns the specified iterator with the specified function applied to each item |
| [max()](https://www.w3schools.com/python/ref_func_max.asp) | Returns the largest item in an iterable |
| [memoryview()](https://www.w3schools.com/python/ref_func_memoryview.asp) | Returns a memory view object |
| [min()](https://www.w3schools.com/python/ref_func_min.asp) | Returns the smallest item in an iterable |
| [next()](https://www.w3schools.com/python/ref_func_next.asp) | Returns the next item in an iterable |
| [object()](https://www.w3schools.com/python/ref_func_object.asp) | Returns a new object |
| [oct()](https://www.w3schools.com/python/ref_func_oct.asp) | Converts a number into an octal |
| [open()](https://www.w3schools.com/python/ref_func_open.asp) | Opens a file and returns a file object |
| [ord()](https://www.w3schools.com/python/ref_func_ord.asp) | Convert an integer representing the Unicode of the specified character |
| [pow()](https://www.w3schools.com/python/ref_func_pow.asp) | Returns the value of x to the power of y |
| [print()](https://www.w3schools.com/python/ref_func_print.asp) | Prints to the standard output device |
| property() | Gets, sets, deletes a property |
| [range()](https://www.w3schools.com/python/ref_func_range.asp) | Returns a sequence of numbers, starting from 0 and increments by 1 (by default) |
| repr() | Returns a readable version of an object |
| [reversed()](https://www.w3schools.com/python/ref_func_reversed.asp) | Returns a reversed iterator |
| [round()](https://www.w3schools.com/python/ref_func_round.asp) | Rounds a numbers |
| [set()](https://www.w3schools.com/python/ref_func_set.asp) | Returns a new set object |
| [setattr()](https://www.w3schools.com/python/ref_func_setattr.asp) | Sets an attribute (property/method) of an object |
| [slice()](https://www.w3schools.com/python/ref_func_slice.asp) | Returns a slice object |
| [sorted()](https://www.w3schools.com/python/ref_func_sorted.asp) | Returns a sorted list |
| staticmethod() | Converts a method into a static method |
| [str()](https://www.w3schools.com/python/ref_func_str.asp) | Returns a string object |
| [sum()](https://www.w3schools.com/python/ref_func_sum.asp) | Sums the items of an iterator |
| [super()](https://www.w3schools.com/python/ref_func_super.asp) | Returns an object that represents the parent class |
| [tuple()](https://www.w3schools.com/python/ref_func_tuple.asp) | Returns a tuple |
| [type()](https://www.w3schools.com/python/ref_func_type.asp) | Returns the type of an object |
| [vars()](https://www.w3schools.com/python/ref_func_vars.asp) | Returns the \_\_dict\_\_ property of an object |
| [zip()](https://www.w3schools.com/python/ref_func_zip.asp) | Returns an iterator, from two or more iterators |

* Python Numbers

In Python, numbers are a core data-type essential for performing arithmetic operations and calculations. [Python](https://www.geeksforgeeks.org/python-programming-language/) supports three types of numbers, including integers, floating-point numbers and complex numbers. Here's an overview of each:

a = 4 # integer

b = 4.5 # float

c = 4j # complex number

​

print(type(a))

print(type(b))

print(type(c))

**Output**

<class 'int'>

<class 'float'>

<class 'complex'>

Let's understand python number and it's types :

* [Python Integer](https://www.geeksforgeeks.org/python-numbers/#python-integer)
* [Python Float](https://www.geeksforgeeks.org/python-numbers/#python-float-type)
* [Python Complex](https://www.geeksforgeeks.org/python-numbers/#python-complex-type)
* [Type Conversion in Python](https://www.geeksforgeeks.org/python-numbers/#type-conversion-in-python)

1. **Python Integer**

**Python int**is the whole [number](https://www.geeksforgeeks.org/numbers/), including negative numbers but not fractions. In Python, there is no limit to how long an integer value can be.

Example:

x = 5 *# A positive integer*

y = -23 *# A negative integer*

z = 0 *# Zero is also considered an integer*

**Performing arithmetic Operations on int type:**

To check the result of arithmetic operations we can simply print the output using print(res). To check the type of the object, we can use **print(type(res))**.

# Addition

res = 5 + 3 # Output: 8

​

# Subtraction

res = 10 - 4 # Output: 6

​

# Multiplication

res = 7 \* 6 # Output: 42

​

# Division

res = 15 / 4 # Output: 3.75

​

# Floor Division

res = 15 // 4 # Output: 3

​

# Modulus (%)

res = 15 % 4 # Output: 3 (because 15 divided by 4 gives remainder 3)

​

# Exponentiation (\*\*)

res = 2 \*\* 3 # Output: 8 (because 2 raised to the power of 3 is 8)

​

# Absolute Value (abs)

res = abs(-10) # Output: 10 (absolute value of -10)

​

# Round (round)

res = round(3.14159, 2) # Output: 3.14 (rounds to 2 decimal places)

1. **Python Float**

This is a real [number](https://www.geeksforgeeks.org/numbers/)with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation. . Some examples of numbers that are represented as floats are 0.5 and -7.823457.

They can be created directly by entering a number with a decimal point, or by using operations such as division on integers. Extra zeros present at the number's end are ignored automatically.

**Examples:**

a = 3.14 # A positive float

b = -0.99 # A negative float

c = 0.0 # A float value that represents zero

**Performing arithmetic Operations on float type:**

To check the result of arithmetic operations, we can simply print the output using print(res). To check the type of the object, we can use print(type(res)).

# Addition (float)

res = 3.5 + 2.2 # Output: 5.7

​

# Subtraction (float)

res = 7.8 - 4.3 # Output: 3.5

​

# Multiplication (float)

res = 5.5 \* 2.0 # Output: 11.0

​

# Division (float)

res = 9.0 / 4.5 # Output: 2.0

​

# Floor Division (float)

res = 9.0 // 4.5 # Output: 2.0 (it returns the truncated quotient)

​

# Modulus (%) (float)

res = 9.0 % 4.5 # Output: 0.0 (remainder when divided)

​

# Exponentiation (float)

res = 2.5 \*\* 2 # Output: 6.25 (2.5 raised to the power of 2)

​

# Absolute Value (float)

res = abs(-7.4) # Output: 7.4 (absolute value)

​

# Round (float)

res = round(3.14159, 2) # Output: 3.14 (rounds to 2 decimal places)

The accuracy of a floating-point number is only up to 15 decimal places, the 16th place can be inaccurate.

1. **Python Complex**

A complex number is a number that consists of real and imaginary parts. **For example**, 2 + 3j is a complex number where 2 is the real component, and 3 multiplied by j is an imaginary part.

**Performing arithmetic operations on complex type:**

To check the result of arithmetic operations, we can simply print the output using print(res). To check the type of the object, we can use print(type(res)).

# Addition (complex)

res = (3 + 4j) + (1 + 2j) # Output: (4 + 6j)

​

# Subtraction (complex)

res = (5 + 6j) - (2 + 3j) # Output: (3 + 3j)

​

# Multiplication (complex)

res = (2 + 3j) \* (1 + 4j) # Output: (-10 + 11j)

​

# Division (complex)

res = (8 + 6j) / (2 + 3j) # Output: (2.0 + 0.0j)

​

# Exponentiation (complex)

res = (1 + 1j) \*\* 2 # Output: (0 + 2j) (raises the complex number to the power of 2)

​

# Absolute Value (complex)

res = abs(3 + 4j) # Output: 5.0 (absolute value of a complex number, which is sqrt(3^2 + 4^2))

​

# Conjugate of a complex number

res = (3 + 4j).conjugate() # Output: (3 - 4j) (negates the imaginary part)

​

# Real and Imaginary parts of a complex number

real = (3 + 4j).real # Output: 3.0

imag = (3 + 4j).imag # Output: 4.0

* Type Conversion in Python

We can convert one number into the other form by two methods:

**Using built-in functions and Arithmetic Operations**

We can also use built-in functions like int(), float() and complex() to convert into different types explicitly. We can also use operations like addition and subtraction to change the type of number implicitly(automatically), if one of the operands is float. This method is not working for complex numbers.

#Using Built-In Function

a = 2

print(float(a))

​

b = 5.6

print(int(b))

​

c = '3'

print(type(int(c)))

​

d = '5.6'

print(type(float(c)))

​

e = 5

print(complex(e))

​

f = 6.5

print(complex(f))

​

#Using Arithmetic Operations:

a = 1.6

b = 5

​

c = a + b

​

print(c)

**Output**

2.0

5

<class 'int'>

<class 'float'>

(5+0j)

(6.5+0j)

6.6

When we convert float to int, the decimal part is truncated.

* **Built-in Functions for Numbers in Python**

These functions are built into Python and can be used **without importing any module**.

| **Function** | **Description** | **Example** | **Output** |
| --- | --- | --- | --- |
| abs(x) | Returns the **absolute value** of a number. | abs(-7) | 7 |
| round(x[, n]) | Rounds a number x to n digits after the decimal (optional). | round(3.14159, 2) | 3.14 |
| pow(x, y[, z]) | Returns x to the power of y, optionally modulo z. | pow(2, 3) pow(2, 3, 5) | 8 3 |
| divmod(x, y) | Returns a **tuple** of quotient and remainder: (x // y, x % y). | divmod(10, 3) | (3, 1) |
| max(...) | Returns the **largest value** among inputs or in an iterable. | max(1, 5, 3) max([1, 7, 2]) | 5 7 |
| min(...) | Returns the **smallest value** among inputs or in an iterable. | min(1, 5, 3) min([1, 7, 2]) | 1 1 |
| sum(iterable[, start]) | Sums all items in the iterable. Optional start value is added to the result. | sum([1, 2, 3]) sum([1, 2], 10) | 6 13 |
| int(x[, base]) | Converts a string or number to an **integer**. Optional base (2 to 36). | int("101", 2) | 5 |
| float(x) | Converts a string or number to a **floating point number**. | float("3.14") | 3.14 |
| complex(x[, y]) | Creates a **complex number** with real part x and imaginary part y. | complex(2, 3) | (2+3j) |

**🔍 Example Code:**

print(abs(-9))

print(round(3.567, 2))

print(pow(3, 2))

print(pow(3, 2, 4))

print(divmod(13, 4))

print(max(1, 4, 2))

print(min(5, 0, -1))

print(sum([1, 2, 3]))

print(int("10", 2))

print(float("7.5"))

print(complex(4, 5))

output:

9

3.57

9

1

(3, 1)

4

-1

6

2

7.5

(4+5j)

* Number-Related Modules in Python

Python provides several **standard modules** for performing advanced operations with numbers. These modules help you handle:

* Mathematical computations
* Random number generation
* High-precision decimal arithmetic
* Fractions and rational numbers
* Statistical calculations

|  |  |  |
| --- | --- | --- |
| Module | Purpose | Common Uses |
| math | Mathematical operations | sqrt, factorial, log, trigonometry |
| random | Random number generation | Simulations, games, tests |
| decimal | High-precision decimal numbers | Financial, scientific precision |
| fractions | Rational number arithmetic | Exact fractions |
| statistics | Statistical calculations | Mean, median, mode, standard deviation |

**1. math Module**

Used for performing **mathematical operations** like square roots, trigonometry, logarithms, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Description | Example | Output |
| math.sqrt(x) | Square root | math.sqrt(25) | 5.0 |
| math.pow(x, y) | Power (float) | math.pow(2, 3) | 8.0 |
| math.factorial(x) | Factorial | math.factorial(5) | 120 |
| math.floor(x) | Round down | math.floor(3.9) | 3 |
| math.ceil(x) | Round up | math.ceil(3.1) | 4 |
| math.gcd(x, y) | Greatest common divisor | math.gcd(12, 15) | 3 |

✅ **Import**: import math

**2. random Module**

Used for **random number generation**, simulations, and games.

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Description | Example | Output |
| random.random() | Random float [0,1) | random.random() | e.g. 0.74 |
| random.randint(a, b) | Random int in [a, b] | random.randint(1, 6) | e.g. 4 |
| random.choice(seq) | Random item from list | random.choice([1,2,3]) | e.g. 2 |
| random.shuffle(seq) | Shuffle the list | random.shuffle(my\_list) | Shuffled list |

✅ **Import**: import random

**3. decimal Module**

Used for **precise floating-point arithmetic**, especially for financial applications.

**Key Concepts:**

* Avoids floating-point rounding errors.
* Precision can be set manually.

**Example:**

from decimal import Decimal, getcontext

getcontext().prec = 4

x = Decimal('0.1') + Decimal('0.2')

print(x)

output: 0.3

✅ **Import**: from decimal import Decimal, getcontext

**4. fractions Module**

Handles **fractions and rational numbers** exactly.

|  |  |  |
| --- | --- | --- |
| Function | Description | Output |
| Fraction(1, 2) | Creates 1/2 | Fraction(1, 2) |
| Fraction('0.5') | From decimal string | Fraction(1, 2) |

**Example:**

from fractions import Fraction

f=Fraction(1, 3) + Fraction(1, 6)

print(f)

output: 1/2

✅ **Import**: from fractions import Fraction

**5. statistics Module**

Provides functions for **statistical calculations**.

**Key Functions:**

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Description | Example | Output |
| mean(data) | Average value | mean([1,2,3]) | 2.0 |
| median(data) | Middle value | median([1,2,3]) | 2 |
| mode(data) | Most frequent | mode([1,1,2]) | 1 |
| stdev(data) | Std. deviation | stdev([1,2,3,4,5]) | 1.58... |

✅ **Import**: from statistics import mean, median, mode, stdev

* Python Strings

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

We use single quotes or double quotes to represent a string in Python. For example,

# create a string using double quotes

string1 = "Python programming"

# create a string using single quotes

string1 = 'Python programming'

Here, we have created a string [variable](https://www.programiz.com/python-programming/variables-constants-literals) named string1. The variable is initialized with the string "Python Programming".

**Example: Python String**

# create string type variables

name = "Python"

print(name)

message = "Welcome to Python."

print(message)

**Output**

Python

Welcome to Python.

In the above example, we have created string-type variables: name and message with values "Python" and "Welcome to Python" respectively.

Here, we have used double quotes to represent strings, but we can use single quotes too.

**Access String Characters in Python**

We can access the characters in a string in three ways.

* **Indexing:** One way is to treat strings as a [list](https://www.programiz.com/python-programming/list) and use index values. For example,

str = 'hello'

# access 1st index element

print(str[1]) # "e"

[Run Code](https://www.programiz.com/python-programming/online-compiler)

* **Negative Indexing**: Similar to a list, Python allows negative indexing for its strings. For example,

str = 'hello'

# access 1st index element

print(str[-4]) # "e"

[Run Code](https://www.programiz.com/python-programming/online-compiler)

* **Slicing:** Access a range of characters in a string by using the slicing operator colon :. For example,

str = 'hello'

# access 1st index element

print(str[1:4]) # "ell"

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Python Strings are Immutable**

In Python, strings are immutable. That means the characters of a string cannot be changed. For example,

message = 'hello ram'

message[0] = 'H'

print(message)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

TypeError: 'str' object does not support item assignment

**Python Multiline String**

We can also create a multiline string in Python. For this, we use triple double quotes """ or triple single quotes '''. For example,

# multiline string

message = """

Never gonna give you up

Never gonna let you down

"""

print(message)

**Output**

Never gonna give you up

Never gonna let you down

In the above example, anything inside the enclosing triple quotes is one multiline string.

**Python String Operations**

Many operations can be performed with strings, which makes it one of the most used [data types](https://www.programiz.com/python-programming/variables-datatypes) in Python.

**1. Compare Two Strings**

We use the == operator to compare two strings. If two strings are equal, the operator returns True. Otherwise, it returns False. For example,

str1 = "Hello, world!"

str2 = "Hello, Python."

str3 = "Hello, world!"

# compare str1 and str2

print(str1 == str2)

# compare str1 and str3

print(str1 == str3)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

False

True

In the above example,

1. str1 and str2 are not equal. Hence, the result is False.
2. str1 and str3 are equal. Hence, the result is True.

**2. Join Two or More Strings**

In Python, we can join (concatenate) two or more strings using the + operator.

greet = "Hello, "

name = "Ram"

# using + operator

result = greet + name

print(result)

**Output**

Hello, Ram

[Run Code](https://www.programiz.com/python-programming/online-compiler)

In the above example, we have used the + operator to join two strings: greet and name.

**Iterate Through a Python String**

We can iterate through a string using a [for loop](https://www.programiz.com/python-programming/for-loop). For example,

greet = 'Hello'

# iterating through greet string

for letter in greet:

    print(letter)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

H

e

l

l

o

**Python String Length**

In Python, we use the [len()](https://www.programiz.com/python-programming/methods/built-in/len) method to find the length of a string. For example,

greet = 'Hello'

# count length of greet string

print(len(greet))

output: 5

**String Membership Test**

We can test if a substring exists within a string or not, using the keyword in.

[print('a' in 'program')](https://www.programiz.com/python-programming/online-compiler" \t "_blank)

[print('at' not in 'battle')](https://www.programiz.com/python-programming/online-compiler" \t "_blank)

output:

True

False

**Methods of Python String**

Besides those mentioned above, there are various [string methods](https://www.programiz.com/python-programming/methods/string) present in Python. Here are some of those methods:

|  |  |
| --- | --- |
| Methods | Description |
| [upper()](https://www.programiz.com/python-programming/methods/string/upper) | Converts the string to uppercase |
| [lower()](https://www.programiz.com/python-programming/methods/string/lower) | Converts the string to lowercase |
| [partition()](https://www.programiz.com/python-programming/methods/string/partition) | Returns a tuple |
| [replace()](https://www.programiz.com/python-programming/methods/string/replace) | Replaces substring inside |
| [find()](https://www.programiz.com/python-programming/methods/string/find) | Returns the index of the first occurrence of substring |
| [rstrip()](https://www.programiz.com/python-programming/methods/string/rstrip) | Removes trailing characters |
| [split()](https://www.programiz.com/python-programming/methods/string/split) | Splits string from left |
| [startswith()](https://www.programiz.com/python-programming/methods/string/startswith) | Checks if string starts with the specified string |
| [isnumeric()](https://www.programiz.com/python-programming/methods/string/isnumeric) | Checks numeric characters |
| [index()](https://www.programiz.com/python-programming/methods/string/index) | Returns index of substring |

**Python String Formatting (f-Strings)**

Python [f-Strings](https://www.programiz.com/python-programming/string-interpolation#:~:text=f%2Dstrings,Python%20expressions%20inside%20string%20constants.) makes it easy to print values and variables. For example,

name = 'Cathy'

country = 'UK'

print(f'{name} is from {country}')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Cathy is from UK

Here, f'{name} is from {country}' is an **f-string**.

This new formatting syntax is powerful and easy to use. From now on, we will use f-Strings to print strings and variables.

* Python List

In Python, lists allow us to store multiple items in a single variable. For example, if you need to store the ages of all the students in a class, you can do this task using a list.

Lists are similar to arrays (dynamic arrays that allow us to store items of different data types) in other programming languages.

**Create a Python List**

We create a list by placing elements inside square brackets [], separated by commas. For example,

# a list of three elements

ages = [19, 26, 29]

print(ages)

**Output**

[19, 26, 29]

1. **List Items of Different Types**

Python lists are very flexible. We can also store data of different data types in a list. For example,

# a list containing strings, numbers and another list

student = ['Jack', 32, 'Computer Science', [2, 4]]

print(student)

# an empty list

empty\_list = []

print(empty\_list)

output:

['Jack', 32, 'Computer Science', [2, 4]]

[]

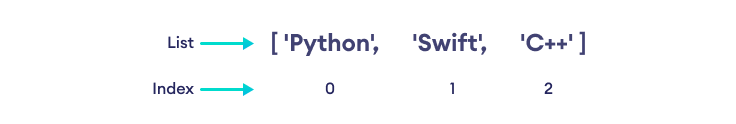
**List Characteristics**

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.

1. **Access List Elements**

Each element in a list is associated with a number, known as an **index**. The index of first item is **0**, the index of second item is **1**, and so on.

Index of List Elements

We use these indices to access items of a list. For example,

languages = ['Python', 'Swift', 'C++']

# access the first element

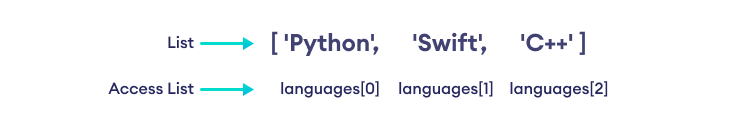
print('languages[0] =', languages[0])

# access the third element

print('languages[2] =', languages[2])

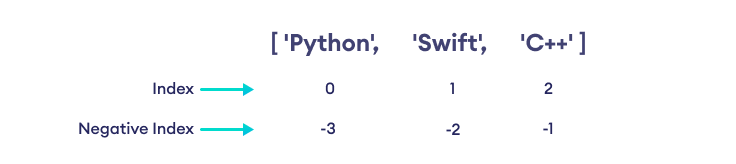
**Output**

languages[0] = Python

languages[2] = C++

1. **Negative Indexing**

In Python, a list can also have negative indices. The index of the last element is **-1**, the second last element is **-2** and so on.



Let's see an example.

languages = ['Python', 'Swift', 'C++']

# access the last item

print('languages[-1] =', languages[-1])

# access the third last item

print('languages[-3] =', languages[-3])

**Output**

languages[-1] = C++

languages[-3] = Python

1. **Slicing of a List in Python**

If we need to access a portion of a list, we can use the slicing operator, :. For example,

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

print("my\_list =", my\_list)

print("my\_list[2: 5] =", my\_list[2: 5])

print("my\_list[2: -2] =", my\_list[2: -2])

print("my\_list[0: 3] =", my\_list[0: 3])

**Output**

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']

1. **Omitting Start and End Indices in Slicing**

If you omit the start index, the slicing starts from the first element. Similarly, if you omit the last index, the slicing ends at the last element. For example,

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[:])

output:

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']

1. **Add Elements to a Python List**

As mentioned earlier, lists are mutable and we can change items of a list. To add an item to the end of a list, we can use the list [append() method](https://www.programiz.com/python-programming/methods/list/append). For example,

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

print('Original List:', fruits)

fruits.append('cherry')

print('Updated List:', fruits)

**Output**

Original List: ['apple', 'banana', 'orange']

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

1. **Add Elements at the Specified Index**

We can insert an element at the specified index to a list using the [insert()](https://www.programiz.com/python-programming/methods/list/insert) method. For example,

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

print("Original List:", fruits)

fruits.insert(2, 'cherry')

print("Updated List:", fruits)

**Output**

Original List: ['apple', 'banana', 'orange']

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

1. **Add Elements to a List From Other Iterables**

The list [extend() method](https://www.programiz.com/python-programming/methods/list/extend) method all the items of the specified iterable, such as list, tuple, dictionary or string , to the end of a list. For example,

numbers = [1, 3, 5]

print('Numbers:', numbers)

even\_numbers  = [2, 4, 6]

print('Even numbers:', numbers)

# adding elements of one list to another

numbers.extend(even\_numbers)

print('Updated Numbers:', numbers)

**Output**

Numbers: [1, 3, 5]

Even numbers: [1, 3, 5]

Updated Numbers: [1, 3, 5, 2, 4, 6]

1. **Change List Items**

We can change the items of a list by assigning new values using the = operator. For example,

colors = ['Red', 'Black', 'Green']

print('Original List:', colors)

# change the first item to 'Purple'

colors[2] = 'Purple'

# change the third item to 'Blue'

colors[2] = 'Blue'

print('Updated List:', colors)

**Output**

Original List: ['Red', 'Black', 'Green']

Updated List: ['Red', 'Black', 'Blue']

Here, we have replaced

* the element at index **0** to 'Purple'
* the element at index **2** to 'Blue'

1. **Remove an Item From a List**

We can remove the specified item from a list using the [remove()](https://www.programiz.com/python-programming/methods/list/remove) method. For example,

numbers = [2,4,7,9]

# remove 4 from the list

numbers.remove(4)

print(numbers)

**Output**

[2, 7, 9]

1. **Remove One or More Elements of a List**

Instead of using the remove() method, we can delete an item from a list using the [del statement](https://www.programiz.com/python-programming/del). The del statement can also be used to delete multiple elements or even the entire list.

names = ['John', 'Eva', 'Laura', 'Nick', 'Jack']

# delete the item at index 1

del names[1]

print(names)

# delete items from index 1 to index 2

del names[1: 3]

print(names)

# delete the entire list

del names

# Error! List doesn't exist.

print(names)

**Output**

['John', 'Laura', 'Nick', 'Jack']

['John', 'Jack']

Traceback (most recent call last):

File "c:\durgaprasad\projects\main.py", line 15, in <module>

print(names)

^^^^^

NameError: name 'names' is not defined

1. **Python List Length**

To find the number of elements (length) of a list, we can use the built-in [len() function](https://www.programiz.com/python-programming/methods/built-in/len). For example,

cars = ['BMW', 'Mercedes', 'Tesla']

print('Total Elements:', len(cars))

**Output**

Total Elements: 3

1. **Iterating Through a List**

We can use a [for loop](https://www.programiz.com/python-programming/for-loop) to iterate over the elements of a list. For example,

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

# iterate through the list

for fruit in fruits:

    print(fruit)

**Output**

apple

banana

orange

**Python List Methods**

Python has many useful [list methods](https://www.programiz.com/python-programming/methods/list) that make it really easy to work with lists.

|  |  |
| --- | --- |
| Method | Description |
| [append()](https://www.programiz.com/python-programming/methods/list/append) | Adds an item to the end of the list |
| [extend()](https://www.programiz.com/python-programming/methods/list/extend) | Adds items of lists and other iterables to the end of the list |
| [insert()](https://www.programiz.com/python-programming/methods/list/insert) | Inserts an item at the specified index |
| [remove()](https://www.programiz.com/python-programming/methods/list/remove) | Removes the specified value from the list |
| [pop()](https://www.programiz.com/python-programming/methods/list/pop) | Returns and removes item present at the given index |
| [clear()](https://www.programiz.com/python-programming/methods/list/clear) | Removes all items from the list |
| [index()](https://www.programiz.com/python-programming/methods/list/index) | Returns the index of the first matched item |
| [count()](https://www.programiz.com/python-programming/methods/list/count) | Returns the count of the specified item in the list |
| [sort()](https://www.programiz.com/python-programming/methods/list/sort) | Sorts the list in ascending/descending order |
| [reverse()](https://www.programiz.com/python-programming/methods/list/reverse) | Reverses the item of the list |
| [copy()](https://www.programiz.com/python-programming/methods/list/copy) | Returns the shallow copy of the list |

* Python Tuple

A tuple is a collection similar to a [Python list](https://www.programiz.com/python-programming/list). The primary difference is that we cannot modify a tuple once it is created.

**Create a Python Tuple**

We create a tuple by placing items inside parentheses (). For example,

numbers = (1, 2, -5)

print(numbers)

**More on Tuple Creation**

Create a Tuple Using tuple() Constructor

Different Types of Python Tuples

**Tuple Characteristics**

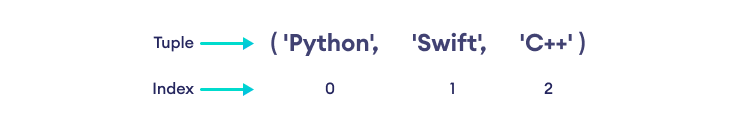
Tuples are:

* **Ordered** - They maintain the order of elements.
* **Immutable** - They cannot be changed after creation.
* **Allow duplicates** - They can contain duplicate values.

1. **Access Tuple Items**

Each item in a tuple is associated with a number, known as a **index**.

The index always starts from **0**, meaning the first item of a tuple is at index **0**, the second item is at index **1,** and so on.



Index of Tuple Item

1. **Access Items Using Index**

We use index numbers to access tuple items. For example,

languages = ('Python', 'Swift', 'C++')

# access the first item

print(languages[0])   # Python

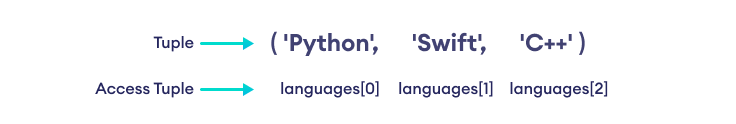
# access the third item

print(languages[2])   # C++

output:

Python

C++



Access Tuple Items

1. **Tuple Cannot be Modified**

Python tuples are immutable (unchangeable). We cannot add, change, or delete items of a tuple.

If we try to modify a tuple, we will get an error. For example,

cars = ('BMW', 'Tesla', 'Ford', 'Toyota')

# trying to modify a tuple

cars[0] = 'Nissan'    # error

print(cars)

output:

Traceback (most recent call last):

File "c:\durgaprasad\projects\main.py", line 4, in <module>

cars[0] = 'Nissan' # error

~~~~^^^

TypeError: 'tuple' object does not support item assignment

1. **Python Tuple Length**

We use the [len()](https://www.programiz.com/python-programming/methods/built-in/len) function to find the number of items present in a tuple. For example,

cars = ('BMW', 'Tesla', 'Ford', 'Toyota')

print('Total Items:', len(cars))

  output:

   Total Items: 4

1. **Iterate Through a Tuple**

We use the [for loop](https://www.programiz.com/python-programming/for-loop) to iterate over the items of a tuple. For example,

fruits = ('apple','banana','orange')

# iterate through the tuple

for fruit in fruits:

    print(fruit)

**Output**

apple

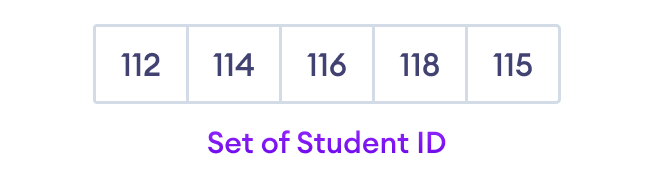
banana

orange

* Python Sets

A set is a collection of unique data, meaning that elements within a set cannot be duplicated.

For instance, if we need to store information about student IDs, a set is suitable since student IDs cannot have duplicates.



Python Set Elements

1. **Create a Set in Python**

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

A set can have any number of items and they may be of different types (integer, float, [tuple](https://www.programiz.com/python-programming/tuple), [string](https://www.programiz.com/python-programming/string), etc.). But a set cannot have mutable elements like [lists](https://www.programiz.com/python-programming/list), sets or [dictionaries](https://www.programiz.com/python-programming/dictionary) as its elements.

Let's see an example,

# create a set of integer type

student\_id = {112, 114, 116, 118, 115}

print('Student ID:', student\_id)

# create a set of string type

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

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

# create a set of mixed data types

mixed\_set = {'Hello', 101, -2, 'Bye'}

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

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Student ID: {112, 114, 115, 116, 118}

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

Set of mixed data types: {'Hello', 'Bye', 101, -2}

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.

1. **Duplicate Items in a Set**

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

numbers = {2, 4, 6, 6, 2, 8}

print(numbers)

output:

{8, 2, 4, 6}

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

1. **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. The set data type does not support it.

**Add Items to a Set in Python**

In Python, we use the [add()](https://www.programiz.com/python-programming/methods/set/add) method to add an item to a set. For example,

numbers = {21, 34, 54, 12}

print('Initial Set:',numbers)

# using add() method

numbers.add(32)

print('Updated Set:', numbers)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Initial Set: {34, 12, 21, 54}

Updated Set: {32, 34, 12, 21, 54}

**Update Python Set**

The [update()](https://www.programiz.com/python-programming/methods/set/update) method is used to update the set with items other collection types (lists, tuples, sets, etc). For example,

companies = {'Lacoste', 'Ralph Lauren'}

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

# using update() method

companies.update(tech\_companies)

print(companies)

output:

{'google', 'apple', 'Ralph Lauren', 'Lacoste'}

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Here, all the unique elements of tech\_companies are added to the companies set.

1. **Remove an Element from a Set**

We use the [discard()](https://www.programiz.com/python-programming/methods/set/discard) method to remove the specified element from a set. For example,

languages = {'Swift', 'Java', 'Python'}

print('Initial Set:',languages)

# remove 'Java' from a set

removedValue = languages.discard('Java')

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

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Initial Set: {'Python', 'Swift', 'Java'}

Set after remove(): {'Python', 'Swift'}

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

**Built-in Functions with Set**

Here are some of the popular built-in functions that allow us to perform different operations on a set.

|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Returns True if all elements of the set are true (or if the set is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Returns True if any element of the set is true. If the set is empty, returns False. |
| [enumerate()](https://www.programiz.com/python-programming/methods/built-in/enumerate) | Returns an enumerate object. It contains the index and value for all the items of the set as a pair. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Returns the length (the number of items) in the set. |
| [max()](https://www.programiz.com/python-programming/methods/built-in/max) | Returns the largest item in the set. |
| [min()](https://www.programiz.com/python-programming/methods/built-in/min) | Returns the smallest item in the set. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Returns a new sorted list from elements in the set(does not sort the set itself). |
| [sum()](https://www.programiz.com/python-programming/methods/built-in/sum) | Returns the sum of all elements in the set. |

1. **Iterate Over a Set in Python**

fruits = {"Apple", "Peach", "Mango"}

# for loop to access each fruits

for fruit in fruits:

    print(fruit)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Mango

Peach

Apple

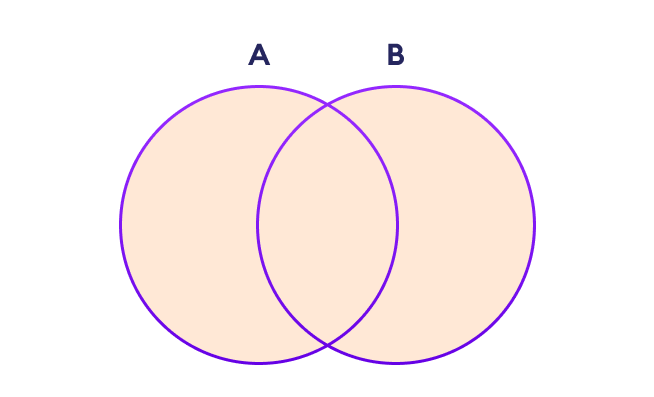
Here, we have used [for loop](https://www.programiz.com/python-programming/for-loop) to iterate over a set in Python.

**Python Set Operations**

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

1. **Union of Two Sets**

The union of two sets A and B includes all the elements of sets A and B.



We use the | operator or the [union()](https://www.programiz.com/python-programming/methods/set/union) method to perform the set union operation. For example,

# first set

A = {1, 3, 5}

# second set

B = {0, 2, 4}

# perform union operation using |

print('Union using |:', A | B)

# perform union operation using union()

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

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

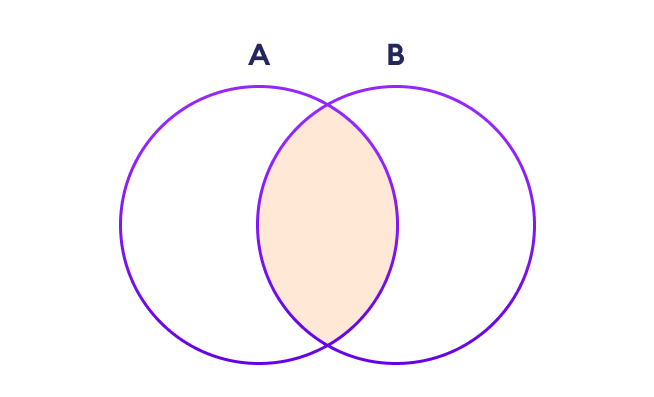
Union using |: {0, 1, 2, 3, 4, 5}

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

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

1. **Set Intersection**

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



In Python, we use the & operator or the [intersection()](https://www.programiz.com/python-programming/methods/set/intersection) method to perform the set intersection operation. For example,

# first set

A = {1, 3, 5}

# second set

B = {1, 2, 3}

# perform intersection operation using &

print('Intersection using &:', A & B)

# perform intersection operation using intersection()

print('Intersection using intersection():', A.intersection(B))

**Output**

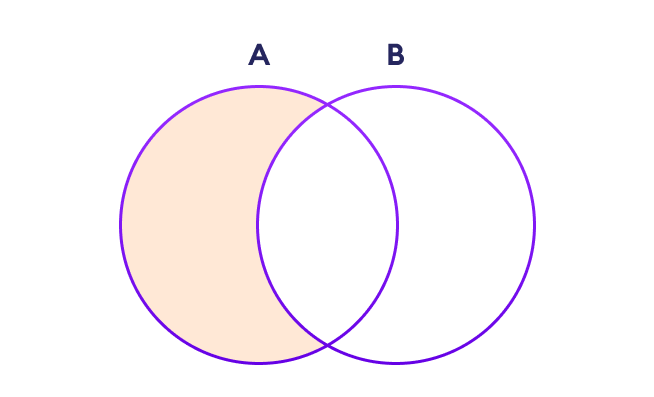
Intersection using &: {1, 3}

Intersection using intersection(): {1, 3}

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

1. **Difference between Two Sets**

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



We use the - operator or the [difference()](https://www.programiz.com/python-programming/methods/set/difference) method to perform the difference between two sets. For example,

# first set

A = {2, 3, 5}

# second set

B = {1, 2, 6}

# perform difference operation using &

print('Difference using &:', A - B)

# perform difference operation using difference()

print('Difference using difference():', A.difference(B))

**Output**

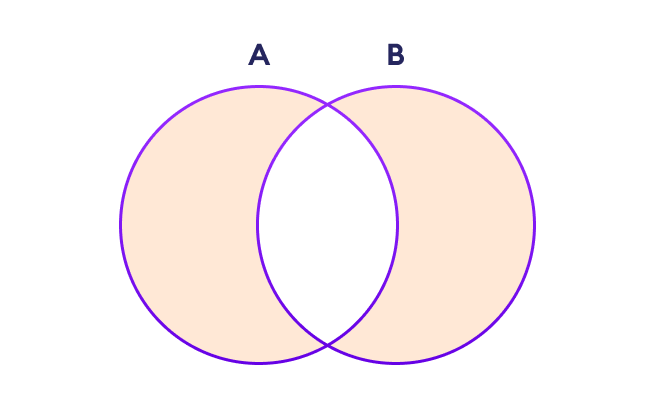
Difference using &: {3, 5}

Difference using difference(): {3, 5}

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

1. **Set Symmetric Difference**

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



In Python, we use the ^ operator or the [symmetric\_difference()](https://www.programiz.com/python-programming/methods/set/symmetric_difference) method to perform symmetric differences between two sets. For example,

# first set

A = {2, 3, 5}

# second set

B = {1, 2, 6}

# perform difference operation using &

print('using ^:', A ^ B)

# using symmetric\_difference()

print('using symmetric\_difference():', A.symmetric\_difference(B))

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

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

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

1. **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:

|  |  |
| --- | --- |
| Method | Description |
| [add()](https://www.programiz.com/python-programming/methods/set/add) | Adds an element to the set |
| [clear()](https://www.programiz.com/python-programming/methods/set/clear) | Removes all elements from the set |
| [copy()](https://www.programiz.com/python-programming/methods/set/copy) | Returns a copy of the set |
| [difference()](https://www.programiz.com/python-programming/methods/set/difference) | Returns the difference of two or more sets as a new set |
| [difference\_update()](https://www.programiz.com/python-programming/methods/set/difference_update) | Removes all elements of another set from this set |
| [discard()](https://www.programiz.com/python-programming/methods/set/discard) | Removes an element from the set if it is a member. (Do nothing if the element is not in set) |
| [intersection()](https://www.programiz.com/python-programming/methods/set/intersection) | Returns the intersection of two sets as a new set |
| [intersection\_update()](https://www.programiz.com/python-programming/methods/set/intersection_update) | Updates the set with the intersection of itself and another |
| [isdisjoint()](https://www.programiz.com/python-programming/methods/set/isdisjoint) | Returns True if two sets have a null intersection |
| [issubset()](https://www.programiz.com/python-programming/methods/set/issubset) | Returns True if another set contains this set |
| [pop()](https://www.programiz.com/python-programming/methods/set/pop) | Removes and returns an arbitrary set element. Raises KeyError if the set is empty |
| [remove()](https://www.programiz.com/python-programming/methods/set/remove) | Removes an element from the set. If the element is not a member, raises a KeyError |
| [symmetric\_difference()](https://www.programiz.com/python-programming/methods/set/symmetric_difference) | Returns the symmetric difference of two sets as a new set |
| [union()](https://www.programiz.com/python-programming/methods/set/union) | Returns the union of sets in a new set |
| [update()](https://www.programiz.com/python-programming/methods/set/update) | Updates the set with the union of itself and others |

* 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).

1. **Create a Dictionary**

We create a dictionary by placing key: value pairs inside curly brackets {}, separated by commas. For example,

# creating a dictionary

country\_capitals = {

  "Germany": "Berlin",

  "Canada": "Ottawa",

  "England": "London"

}

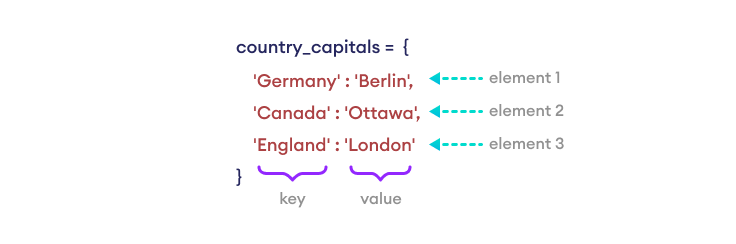
# printing the dictionary

print(country\_capitals)

**Output**

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

The country\_capitals dictionary has three elements (key-value pairs), where 'Germany' is the key and 'Berlin' is the value assigned to it and so on.



Python Dictionary

**Notes**:

* Dictionary keys must be immutable, such as tuples, strings, integers, etc. We cannot use mutable (changeable) objects such as lists as keys.
* We can also create a dictionary using a Python built-in function dict

1. **Access Dictionary Items**

We can access the value of a dictionary item by placing the key inside square brackets.

country\_capitals = {

  "Germany": "Berlin",

  "Canada": "Ottawa",

  "England": "London"

}

# access the value of keys

print(country\_capitals["Germany"])

print(country\_capitals["England"])

**output:**

Berlin

London

1. **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)

**Output**

{'Germany': 'Berlin', 'Canada': 'Ottawa', 'Italy': 'Rome'}

1. **Remove Dictionary Items**

We can use the [del](https://www.programiz.com/python-programming/del) statement to remove an element from a dictionary. For example,

country\_capitals = {

  "Germany": "Berlin",

  "Canada": "Ottawa",

}

# delete item having "Germany" key

del country\_capitals["Germany"]

print(country\_capitals)

**Output**

{'Canada': 'Ottawa'}

**Note**: We can also use the [pop()](https://www.programiz.com/python-programming/methods/dictionary/pop) method to remove an item from a dictionary.

If we need to remove all items from a dictionary at once, we can use the [clear()](https://www.programiz.com/python-programming/methods/dictionary/clear) method.

country\_capitals = {

  "Germany": "Berlin",

  "Canada": "Ottawa",

}

# clear the dictionary

country\_capitals.clear()

print(country\_capitals)

**Output**

{}

1. **Change Dictionary Items**

Python dictionaries are mutable (changeable). We can change the value of a dictionary element by referring to its key. For example,

country\_capitals = {

  "Germany": "Berlin",

  "Italy": "Naples",

  "England": "London"

}

# change the value of "Italy" key to "Rome"

country\_capitals["Italy"] = "Rome"

print(country\_capitals)

**Output**

{'Germany': 'Berlin', 'Italy': 'Rome', 'England': 'London'}

**Note**: We can also use the [update()](https://www.programiz.com/python-programming/methods/dictionary/update) method to add or change dictionary items.

1. **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](https://www.programiz.com/python-programming/for-loop).

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)

**Output**

United States

Italy

Washington D.C.

Rome

**Python Dictionary Methods**

Here are some of the commonly used [dictionary methods](https://www.programiz.com/python-programming/methods/dictionary).

|  |  |
| --- | --- |
| Function | Description |
| [pop()](https://www.programiz.com/python-programming/methods/dictionary/pop) | Removes the item with the specified key. |
| [update()](https://www.programiz.com/python-programming/methods/dictionary/update) | Adds or changes dictionary items. |
| [clear()](https://www.programiz.com/python-programming/methods/dictionary/clear) | Remove all the items from the dictionary. |
| [keys()](https://www.programiz.com/python-programming/methods/dictionary/keys) | Returns all the dictionary's keys. |
| [values()](https://www.programiz.com/python-programming/methods/dictionary/values) | Returns all the dictionary's values. |
| [get()](https://www.programiz.com/python-programming/methods/dictionary/get) | Returns the value of the specified key. |
| [popitem()](https://www.programiz.com/python-programming/methods/dictionary/popitem) | Returns the last inserted key and value as a tuple. |
| [copy()](https://www.programiz.com/python-programming/methods/dictionary/copy) | Returns a copy of the dictionary. |

* Python if...else Statement

In computer programming, the if statement is a conditional statement. It is used to execute a block of code only when a specific condition is met. For example,

Suppose we need to assign different grades to students based on their scores.

1. If a student scores above **90**, assign grade **A**
2. If a student scores above **75**, assign grade **B**
3. If a student scores above **65**, assign grade **C**

These conditional tasks can be achieved using the if statement.

1. **Python if Statement**

An if statement executes a block of code only when the specified condition is met.

**Syntax**

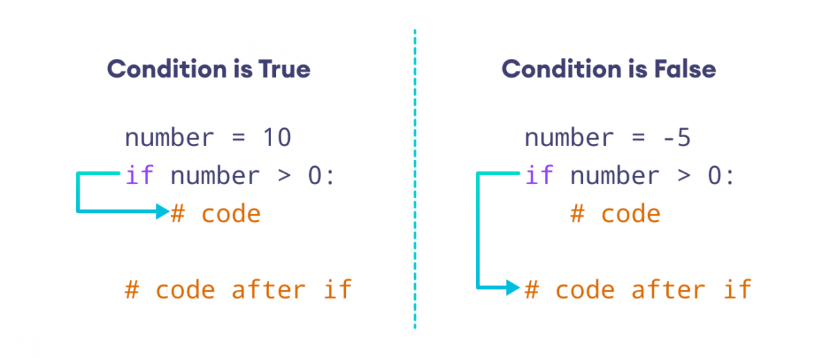
if condition:

    # body of if statement

Here, **condition** is a boolean expression, such as number > 5, that evaluates to either True or False.

* If condition evaluates to True, the body of the if statement is executed.
* If condition evaluates to False, the body of the if statement will be skipped from execution.

Let's look at an example.



**Example: Python if Statement**

number = int(input('Enter a number: '))

# check if number is greater than 0

if number > 0:

    print(f'{number} is a positive number.')

print('A statement outside the if statement.')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Sample Output 1**

Enter a number: 10

10 is a positive number.

A statement outside the if statement.

**Sample Output 2**

Enter a number: -2

A statement outside the if statement.

1. **Python if...else Statement**

An if statement can have an optional else clause. The else statement executes if the condition in the if statement evaluates to False.

**Syntax**

if condition:

    # body of if statement

else:

    # body of else statement

Here, if the condition inside the if statement evaluates to

* **True** - the body of if executes, and the body of else is skipped.
* **False** - the body of else executes, and the body of if is skipped

Let's look at an example.

**Example: Python if…else Statement**

number = int(input('Enter a number: '))

if number > 0:

    print('Positive number')

else:

    print('Not a positive number')

print('This statement always executes')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Sample Output 1**

Enter a number: 10

Positive number

This statement always executes

**Sample Output 2**

Enter a number: 0

Not a positive number

This statement always executes

1. **Python if…elif…else Statement**

The if...else statement is used to execute a block of code among two alternatives.

However, if we need to make a choice between more than two alternatives, we use the if...elif...else statement.

**Syntax**

if condition1:

    # code block 1

elif condition2:

    # code block 2

else:

    # code block 3

Let's look at an example.

**Example: Python if…elif…else Statement**

number = -5

if number > 0:

    print('Positive number')

elif number < 0:

    print('Negative number')

else:

    print('Zero')

print('This statement is always executed')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Negative number

This statement is always executed

1. **Python Nested if Statements**

It is possible to include an if statement inside another if statement. For example,

number = 5

# outer if statement

if number >= 0:

    # inner if statement

    if number == 0:

      print('Number is 0')

    # inner else statement

    else:

        print('Number is positive')

# outer else statement

else:

    print('Number is negative')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Number is positive

* looping statements

Python offers two primary looping statements for repetitive execution of code blocks: for loops and while loops.

1. for Loop:

The for loop is used for iterating over a sequence (such as a list, tuple, string, or range) or other iterable objects. It executes a block of code once for each item in the sequence.

*# Example: Iterating through a list*  
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
 print(fruit)  
  
*# Example: Iterating through a range of numbers*  
for i in range(5): *# Generates numbers from 0 to 4*  
 print(i)

Output:

apple

banana

cherry

0

1

2

3

4

2. while Loop:

The while loop executes a block of code repeatedly as long as a specified condition remains True.

*# Example: Counting up to a certain number*  
count = 0  
while count < 5:  
 print(count)  
 count += 1

Output:

0

1

2

3

4

Loop jumping’s:

* break:

Terminates the loop entirely and transfers control to the statement immediately following the loop.

* continue:

Skips the rest of the current iteration and proceeds to the next iteration of the loop.

* pass:

A null operation; it does nothing. It can be used as a placeholder where a statement is syntactically required but no action is desired.

* Truthiness

In Python, truthiness is a concept that determines how different values are evaluated in boolean contexts. Every object in Python can be tested for its truth value, which helps developers write more concise and expressive code.

**Basic Truthiness Rules**

Python follows a set of fundamental rules when evaluating the truthiness of objects:

|  |  |  |
| --- | --- | --- |
| Value Type | Truthiness | Example |
| None | False | bool(None) == False |
| False | False | bool(False) == False |
| Zero values | False | bool(0) == False |
| Empty sequences | False | bool([]) == False |
| Non-zero numbers | True | bool(42) == True |
| Non-empty sequences | True | bool([1, 2, 3]) == True |

Code Examples

## Demonstrating truthiness

print(bool(0))

print(bool(1))

print(bool([]))

print(bool([1, 2, 3]))

print(bool(None))

**output:**

False

True

False

True

* Python List sort()

The list's sort() method sorts the elements of a list.

**sort() Syntax**

numbers.sort(reverse, key)

The sort() method can take two optional keyword arguments:

* **reverse** - By default False. If True is passed, the list is sorted in descending order.
* **key** - Comparion is based on this function.

**Example**

prime\_numbers = [11, 3, 7, 5, 2]

# sort the list in ascending order

prime\_numbers.sort()

print(prime\_numbers)

output: [2, 3, 5, 7, 11]

**Sort in Descending order**

We can sort a list in descending order by setting reverse to True.

numbers = [7, 3, 11, 2, 5]

# reverse is set to True

numbers.sort(reverse = True)

print(numbers)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

[11, 7, 5, 3, 2]

**Sort a List of Strings**

The sort() method sorts a list of strings in dictionary order.

cities = ["Tokyo", "London", "Washington D.C"]

# sort in dictionary order

cities.sort()

print(f"Dictionary order: {cities}")

# sort in reverse dictionary order

cities.sort(reverse = True)

print(f"Reverse dictionary order: {cities}")

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Dictionary order: ['London', 'Tokyo', 'Washington D.C']

Reverse dictionary order: ['Washington D.C', 'Tokyo', 'London']

* Python List Comprehension

List comprehension offers a concise way to create a new list based on the values of an existing list.

Suppose we have a list of numbers and we desire to create a new list containing the double value of each element in the list.

numbers = [1, 2, 3, 4]

# list comprehension to create new list

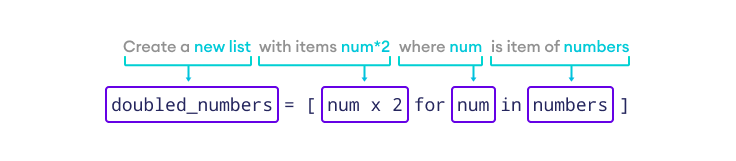
doubled\_numbers = [num \* 2 for num in numbers]

print(doubled\_numbers)

**Output**

[2, 4, 6, 8]

Here is how the list comprehension works:



Python List Comprehension

**Syntax of List Comprehension**

[expression for item in list if condition == True]

Here,

for every item in list, execute the expression if the condition is True.

**Note:** The if statement in list comprehension is optional.

**for Loop vs. List Comprehension**

List comprehension makes the code cleaner and more concise than for loop.

Let's write a program to print the square of each list element using both for loop and list comprehension.

**for Loop**

numbers = [1, 2, 3, 4, 5]

square\_numbers = []

# for loop to square each elements

for num in numbers:

    square\_numbers.append(num \* num)

print(square\_numbers)

Output: [1, 4, 9, 16, 25]

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**List Comprehension**

numbers = [1, 2, 3, 4, 5]

# create a new list using list comprehension

square\_numbers = [num \* num for num in numbers]

print(square\_numbers)

Output: [1, 4, 9, 16, 25]

[Run Code](https://www.programiz.com/python-programming/online-compiler)

It's much easier to understand list comprehension once you know [Python for loop()](https://www.programiz.com/python-programming/for-loop).

* Difference Between Iterator VS Generator

A process that is repeated more than one time by applying the same logic is called an Iteration.  In programming languages like python, a loop is created with few conditions to perform iteration till it exceeds the limit. If the loop is executed 6 times continuously, then we could say the particular block has iterated 6 times.

**Example:**

a = [0, 5, 10, 15, 20]

for i in a:

    if i % 2 == 0:

        print(str(i)+' is an Even Number')

    else:

        print(str(i)+' is an Odd Number')

**Output:**

0 is an Even Number

5 is an Odd Number

10 is an Even Number

15 is an Odd Number

20 is an Even Number

**Iterator**

An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc. Iterators are implemented using a class and a local variable for iterating is not required here, It follows lazy evaluation where the evaluation of the expression will be on hold and stored in the memory until the item is called specifically which helps us to avoid repeated evaluation. As lazy evaluation is implemented, it requires only 1 memory location to process the value and when we are using a large dataset then, wastage of RAM space will be reduced the need to load the entire dataset at the same time will not be there.

Using an iterator-

* **iter()** function is used to create an iterator containing an iterable object.
* **next(**) function is used to call the next element in the iterable object.
* After the iterable object is completed, to use them again reassign them to the same object.

**Example:**

iter\_list = iter(['Python', 'is', 'programming'])

print(next(iter\_list))

print(next(iter\_list))

print(next(iter\_list))

**Output:**

Python

is

programming

**Generators**

It is another way of creating iterators in a simple way where it uses the keyword “yield” instead of returning it in a defined function. Generators are implemented using a function. Just as iterators, generators also follow lazy evaluation. Here, the yield function returns the data without affecting or exiting the function. It will return a sequence of data in an iterable format where we need to iterate over the sequence to use the data as they won’t store the entire sequence in the memory.

**Example:**

def sq\_numbers(n):

    for i in range(1, n+1):

        yield i\*i

a = sq\_numbers(3)

print("The square of numbers 1,2,3 are : ")

print(next(a))

print(next(a))

print(next(a))

**Output:**

The square of numbers 1,2,3 are :

1

4

9