MARUDHAR KESARI JAIN COLLEGE FOR WOMEN, VANIYAMBADI

PG DEPARTMENT OF COMPUTER APPLICATIONS

Subject Name : PYTHON PROGRAMMING

CLASS : I-BCA

SUBJECT CODE :23UCA11

UNIT 1: Basics of Python Programming

History of Python – Features of Python – Literal – Constant – Variable – Identifiers – Keyboard – Build in Data Types – Output Statements – Input Statements – Comments – Indentation – Operators – Expressions – Type Conversions

INTRODUCTION

What is Python

Python is a general-purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is an easy-to-learn yet powerful and versatile scripting language, which makes it attractive for Application Development.

With its interpreted nature, Python's syntax and dynamic typing make it an ideal language for scripting and rapid application development.

Python supports multiple programming patterns, including object-oriented, imperative, and functional or procedural programming styles.

Python is not intended to work in a particular area, such as web programming. It is a multipurpose programming language because it can be used with web, enterprise, 3D CAD, etc.

We don't need to use data types to declare variable because it is dynamically typed, so we can write a=10 to assign an integer value in an integer variable.

Python makes development and debugging fast because no compilation step is included in Python development, and the edit-test-debug cycle is very fast.

Python has many web-based assets, open-source projects, and a vibrant community. Learning the language, working together on projects, and contributing to the Python ecosystem are all made very easy for developers.

Because of its straightforward language framework, Python is easier to understand and write code in. This makes it a fantastic programming language for novices. Additionally, it assists seasoned programmers in writing clearer, error-free code.

Python is an open-source, cost-free programming language. It is utilized in several sectors and disciplines as a result.

In Python, code readability and maintainability are important. As a result, even if the code was developed by someone else, it is easy to understand and adapt by some other developer.

Python has many third-party libraries that can be used to make its functionality easier. These libraries cover many domains, for example, web development, scientific computing, data analysis, and more.

Python Basic Syntax

There is no use of curly braces or semicolon in Python programming language. It is English-like language. But Python uses the indentation to define a block of code. Indentation is nothing but adding whitespace before the statement when it is needed. **For example -**

```
def func():
       statement 1
       statement 2
       …………………
       …………………
         statement N
```

In the above example, the statements that are the same level to the right belong to the function. Generally, we can use four whitespaces to define indentation.

Instead of Semicolon as used in other languages, Python ends its statements with a NewLine character.

Python is a case-sensitive language, which means that uppercase and lowercase letters are treated differently. For example, 'name' and 'Name' are two different variables in Python.

In Python, comments can be added using the '#' symbol. Any text written after the '#' symbol is considered a comment and is ignored by the interpreter. This trick is useful for adding notes to the code or temporarily disabling a code block. It also helps in understanding the code better by some other developers.

'If', 'otherwise', 'for', 'while', 'try', 'except', and 'finally' are a few reserved keywords in Python that cannot be used as variable names. These terms are used in the language for particular reasons and have fixed meanings. If you use these keywords, your code may include errors, or the interpreter may reject them as potential new Variables.

Why learn Python?

Python provides many useful features to the programmer. These features make it the most popular and widely used language. We have listed below few-essential features of Python.

- **Easy to use and Learn:** Python has a simple and easy-to-understand syntax, unlike traditional languages like C, C++, Java, etc., making it easy for beginners to learn.
- **Expressive Language:** It allows programmers to express complex concepts in just a few lines of code or reduces Developer's Time.
- **Interpreted Language:** Python does not require compilation, allowing rapid development and testing. It uses Interpreter instead of Compiler.
- **Object-Oriented Language:** It supports object-oriented programming, making writing reusable and modular code easy.

- **Open Source Language:** Python is open source and free to use, distribute and modify.
- **Extensible:** Python can be extended with modules written in C, C++, or other languages.
- **Learn Standard Library:** Python's standard library contains many modules and functions that can be used for various tasks, such as string manipulation, web programming, and more.
- **GUI Programming Support:** Python provides several GUI frameworks, such as Tkinter and PyQt, allowing developers to create desktop applications easily.
- **Integrated:** Python can easily integrate with other languages and technologies, such as C/C++, Java, and . NET.
- **Embeddable:** Python code can be embedded into other applications as a scripting language.
- **Dynamic Memory Allocation:** Python automatically manages memory allocation, making it easier for developers to write complex programs without worrying about memory management.
- **Wide Range of Libraries and Frameworks:** Python has a vast collection of libraries and frameworks, such as NumPy, Pandas, Django, and Flask, that can be used to solve a wide range of problems.
- **Versatility:** Python is a universal language in various domains such as web development, machine learning, data analysis, scientific computing, and more.
- **Large Community:** Python has a vast and active community of developers contributing to its development and offering support. This makes it easy for beginners to get help and learn from experienced developers.
- **Career Opportunities:** Python is a highly popular language in the job market. Learning Python can open up several career opportunities in data science, artificial intelligence, web development, and more.
- **High Demand:** With the growing demand for automation and digital transformation, the need for Python developers is rising. Many industries seek skilled Python developers to help build their digital infrastructure.
- **Increased Productivity:** Python has a simple syntax and powerful libraries that can help developers write code faster and more efficiently. This can increase productivity and save time for developers and organizations.
- **Big Data and Machine Learning:** Python has become the go-to language for big data and machine learning. Python has become popular among data scientists and machine learning engineers with libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and more.

Where is Python used?

Python is a general-purpose, popular programming language, and it is used in almost every technical field. The various areas of Python use are given below.

- **Data Science:** Data Science is a vast field, and Python is an important language for this field because of its simplicity, ease of use, and availability of powerful data analysis and visualization libraries like NumPy, Pandas, and Matplotlib.

- **Desktop Applications:** PyQt and Tkinter are useful libraries that can be used in GUI - Graphical User Interface-based Desktop Applications. There are better languages for this field, but it can be used with other languages for making Applications.
- **Console-based Applications:** Python is also commonly used to create command-line or console-based applications because of its ease of use and support for advanced features such as input/output redirection and piping.
- **Mobile Applications:** While Python is not commonly used for creating mobile applications, it can still be combined with frameworks like Kivy or BeeWare to create cross-platform mobile applications.
- **Software Development:** Python is considered one of the best software-making languages. Python is easily compatible with both from Small Scale to Large Scale software.
- **Artificial Intelligence:** AI is an emerging Technology, and Python is a perfect language for artificial intelligence and machine learning because of the availability of powerful libraries such as TensorFlow, Keras, and PyTorch.
- **Web Applications:** Python is commonly used in web development on the backend with frameworks like Django and Flask and on the front end with tools like JavaScript and HTML.
- **Enterprise Applications:** Python can be used to develop large-scale enterprise applications with features such as distributed computing, networking, and parallel processing.
- **3D CAD Applications:** Python can be used for 3D computer-aided design (CAD) applications through libraries such as Blender.
- **Machine Learning:** Python is widely used for machine learning due to its simplicity, ease of use, and availability of powerful machine learning libraries.
- **Computer Vision or Image Processing Applications:** Python can be used for computer vision and image processing applications through powerful libraries such as OpenCV and Scikit-image.
- **Speech Recognition:** Python can be used for speech recognition applications through libraries such as SpeechRecognition and PyAudio.
- **Scientific computing:** Libraries like NumPy, SciPy, and Pandas provide advanced numerical computing capabilities for tasks like data analysis, machine learning, and more.
- **Education:** Python's easy-to-learn syntax and availability of many resources make it an ideal language for teaching programming to beginners.
- **Testing:** Python is used for writing automated tests, providing frameworks like unit tests and pytest that help write test cases and generate reports.
- **Gaming:** Python has libraries like Pygame, which provide a platform for developing games using Python.
- **IoT:** Python is used in IoT for developing scripts and applications for devices like Raspberry Pi, Arduino, and others.
- **Networking:** Python is used in networking for developing scripts and applications for network automation, monitoring, and management.

- **DevOps:** Python is widely used in DevOps for automation and scripting of infrastructure management, configuration management, and deployment processes.
- **Finance:** Python has libraries like Pandas, Scikit-learn, and Statsmodels for financial modeling and analysis.
- **Audio and Music:** Python has libraries like Pyaudio, which is used for audio processing, synthesis, and analysis, and Music21, which is used for music analysis and generation.
- **Writing scripts:** Python is used for writing utility scripts to automate tasks like file operations, web scraping, and data processing.

Python Popular Frameworks and Libraries

Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.

- **Web development (Server-side) -** Django Flask, Pyramid, CherryPy
- **GUIs based applications -** Tk, PyGTK, PyQt, PyJs, etc.
- **Machine Learning -** TensorFlow, PyTorch, **Scikit-learn**, Matplotlib, Scipy, etc.
- **Mathematics -** Numpy, Pandas, etc.
- **BeautifulSoup:** a library for web scraping and parsing HTML and XML
- **Requests:** a library for making HTTP requests
- **SQLAlchemy:** a library for working with SQL databases
- **Kivy:** a framework for building multi-touch applications
- **Pygame:** a library for game development
- **Pytest:** a testing framework for Python
- **Django REST framework:** a toolkit for building RESTful APIs
- **FastAPI:** a modern, fast web framework for building APIs
- **Streamlit:** a library for building interactive web apps for machine learning and data science
- **NLTK:** a library for natural language processing

Python Features

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

1) Easy to Learn and Use

Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

2) Expressive Language

Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type **print("Hello World")**. It will take only one line to execute, while Java or C takes multiple lines.

3) Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

5) Free and Open Source

Python is freely available for everyone. It is freely available on its official website www.python.org. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

6) Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

8) Large Standard Library

It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

9) GUI Programming Support

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

10) Integrated

It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes easy to debug the code.

11. Embeddable

The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to **x,** then we don't need to write **int x = 15.** Just write x = 15.

Python History and Versions

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, **Guido Van Rossum** published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
- Python 2.0 added new features such as list comprehensions, garbage collection systems.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.
- ABC programming language is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
- The following programming languages influence Python:
    - ABC language.
    - Modula-3

Why the Name Python?

There is a fact behind choosing the name Python. **Guido van Rossum** was reading the script of a popular BBC comedy series "**Monty Python's Flying Circus**". It was late on-air 1970s.

Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the **"Monty Python's Flying Circus"** for their newly created programming language.

The comedy series was creative and well random. It talks about everything. Thus it is slow and unpredictable, which made it very interesting.

Python is also versatile and widely used in every technical field, such as Machine Learning, Artificial Intelligence, Web Development, Mobile Application, Desktop Application, Scientific Calculation, etc.

Python Version List

Python programming language is being updated regularly with new features and supports. There are lots of update in Python versions, started from 1994 to current release.

A list of Python versions with its released date is given below.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006

Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018
Python 3.8	October 14, 2019

Usage of Python

Python is a general purpose, open source, high-level programming language and also provides number of libraries and frameworks. Python has gained popularity because of its simplicity, easy syntax and user-friendly environment. The usage of Python as follows.

- Desktop Applications
- Web Applications
- Data Science
- Artificial Intelligence
- Machine Learning
- Scientific Computing
- Robotics
- Internet of Things (IoT)

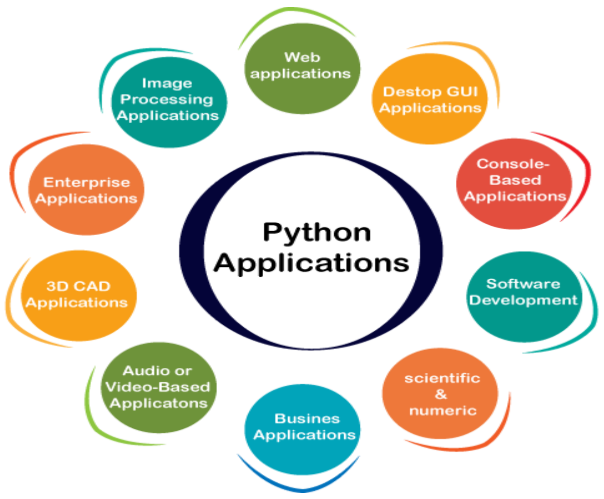
- Gaming
- Mobile Apps
- Data Analysis and Preprocessing

In the next topic, we will discuss the Python Application, where we have defined Python's usage in detail.

Python Applications

Python is known for its general-purpose nature that makes it applicable in almost every domain of software development. Python makes its presence in every emerging field. It is the fastest-growing programming language and can develop any application.

Here, we are specifying application areas where Python can be applied.



1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser, etc. One of Python web-framework named Django is used on **Instagram**. Python provides many useful frameworks, and these are given below:

- Django and Pyramid framework(Use for heavy applications)
- Flask and Bottle (Micro-framework)
- Plone and Django CMS (Advance Content management)

2) Desktop GUI Applications

The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a **Tk GUI library** to develop a user interface. Some popular GUI libraries are given below.

- Tkinter or Tk
- wxWidgetM
- Kivy (used for writing multitouch applications)
- PyQt or Pyside

3) Console-based Application

Console-based applications run from the command-line or shell. These applications are computer program which are used commands to execute. This kind of application was more popular in the old generation of computers. Python can develop this kind of application very effectively. It is famous for having REPL, which means **the Read-Eval-Print Loop** that makes it the most suitable language for the command-line applications.

Python provides many free library or module which helps to build the command-line apps. The necessary **IO** libraries are used to read and write. It helps to parse argument and create console help text out-of-the-box. There are also advance libraries that can develop independent console apps.

4) Software Development

Python is useful for the software development process. It works as a support language and can be used to build control and management, testing, etc.

- **SCons** is used to build control.
- **Buildbot** and **Apache** Gumps are used for automated continuous compilation and testing.
- **Round** or **Trac** for bug tracking and project management.

5) Scientific and Numeric

This is the era of Artificial intelligence where the machine can perform the task the same as the human. Python language is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations.

Implementing machine learning algorithms require complex mathematical calculation. Python has many libraries for scientific and numeric such as Numpy, Pandas, Scipy, Scikit-learn, etc. If you have some basic knowledge of Python, you need to import libraries on the top of the code. Few popular frameworks of machine libraries are given below.

- SciPy
- Scikit-learn
- NumPy
- Pandas
- Matplotlib

6) Business Applications

Business Applications differ from standard applications. E-commerce and ERP are an example of a business application. This kind of application requires extensively, scalability and readability, and Python provides all these features.

Oddo is an example of the all-in-one Python-based application which offers a range of business applications. Python provides a **Tryton** platform which is used to develop the business application.

7) Audio or Video-based Applications

Python is flexible to perform multiple tasks and can be used to create multimedia applications. Some multimedia applications which are made by using Python are **TimPlayer, cplay,** etc. The few multimedia libraries are given below.

- Gstreamer
- Pyglet
- QT Phonon

8) 3D CAD Applications

The CAD (Computer-aided design) is used to design engineering related architecture. It is used to develop the 3D representation of a part of a system. Python can create a 3D CAD application by using the following functionalities.

- Fandango (Popular)
- CAMVOX
- HeeksCNC
- AnyCAD
- RCAM

9) Enterprise Applications

Python can be used to create applications that can be used within an Enterprise or an Organization. Some real-time applications are OpenERP, Tryton, Picalo, etc.

10) Image Processing Application

Python contains many libraries that are used to work with the image. The image can be manipulated according to our requirements. Some libraries of image processing are given below.

- OpenCV
- Pillow
- SimpleITK

Python Variables

- A **Variable** is a location that is named in order to store data while the program is being run.
- In a programming language, **Variables** are words that are used to store values of any data type.

In simple words, when you create a variable, it takes up some memory space based on the value and the type you set to it. The Python interpreter allocates RAM to the variable based on its data type. The variables' values can be altered at any time during the program.
An Identifier is a term used in programming language in order to denote unique name given to these variables.

Syntax

```
variable_name = data values
```

where, variable_name = combination of letters, numbers and an underscore

Note – In Python, we do not specify the data type for the variable. Python automatically understands which data type is being used and allocates memory space accordingly.

Rules to be followed while declaring a Variable name

1. A Variable's name cannot begin with a number. Either an alphabet or the underscore character should be used as the first character.
2. Variable names are case-sensitive and can include alphanumeric letters as well as the underscore character.
3. Variable names cannot contain reserved terms.
4. The equal to sign **'='**, followed by the variable's value, is used to assign variables in Python.

Assigning values to Variables in Python

There are few different methods to assign data elements to a Variable. The most common ones are described below –

1. Simple declaration and assignment of a value to the Variable

In this type, the data values are directly assigned to the Variables in the declaration statement.

Example

```
num = 10
numlist = [1, 3, 5, 7, 9]
```

```
str = 'Hello World'

print(num)
print(numlist)
print(str)
```

Output

```
10
[1, 3, 5, 7, 9]
Hello World
```

Here, we have created 3 variables named as ‘num’, ‘numlist’, and ‘str’. We have assigned all the 3 variables an int value 10, a list of integers, and a string of characters respectively.

2. Changing the value of a Variable

Data values assigned to the variables can be changed at any time. In Layman language, you can think of a Variable as a bag to store items and these items can be replaced at any time.

Example

```
val = 50
print("Initial value:", val)

val = 100  # assigning new value
print("Updated value:", val)
```

Output

```
Initial value: 50
Updated value: 100
```

In the above program, initially the value of Variable ‘val’ was 50. Later it was reassigned to a value of 100.

3. Assign multiple values to multiple Variables

In Python, we can assign multiple values to multiple variables in the same declaration statement by using the following method –

Example

```
name, age, city = 'David', 27, 'New York'

print(name)
print(age)
print(city)
```

Output

```
David
27
New York
```

We can also assign a single value to multiple variables. Let us look at the below example to understand this,

Example

```
a = b = 'Hello'

print('Value of a:', a)
print('Value of b:', b)
```

Output –

```
Value of a: Hello
Value of b: Hello
```

Python Constants

- A Python **Constant** is a variable whose value cannot be changed throughout the program.

Certain values are fixed and are universally proven to be true. These values cannot be changed over time. Such types of values are called as Constants. We can think of Python Constants as a bag full of fruits, but these fruits cannot be removed or changed with other fruits.

Note – Unlike other programming languages, Python does not contain any constants. Instead, Python provides us a Capitalized naming convention method. Any variable written in the Upper case is considered as a Constant in Python.

Rules to be followed while declaring a Constant

1. Python Constants and variable names should contain a combination of lowercase (a-z) or capital (A-Z) characters, numbers (0-9), or an underscore ( ).

2. When using a Constant name, always use UPPERCASE, For example, CONSTANT = 50.
3. The Constant names should not begin with digits.
4. Except for underscore(_), no additional special character (!, #, ^, @, $) is utilized when declaring a constant.
5. We should come up with a catchy name for the python constants. VALUE, for example, makes more sense than V. It simplifies the coding process.

Assigning Values to Constants

Constants are typically declared and assigned in a module in Python. In this case, the module is a new file containing variables, functions, and so on that is imported into the main file. Constants are written in all capital letters with underscores separating the words within the module.

We create a separate file for declaring constants. We then use this file to import the constant module in the main.py file from the other file.

Example

```
# create a separate constant.py file
PI = 3.14
GRAVITY = 9.8



# main.py file
import constant as const

print('Value of PI:', cons.PI)
print('Value of Gravitational force:', cons.GRAVITY)
```

Output

```
Value of PI: 3.14
Value of Gravitational force: 9.8
```

Python Literals

- The data which is being assigned to the variables are called as **Literal**.
- In Python, **Literals** are defined as raw data which is being assigned to the variables or constants.

Let us understand this by looking at a simple example,

```
str = ‘How are you, Sam?’
```

Here, we have declared a variable ‘str’, and the value assigned to it ‘How are you, Sam?’ is a literal of type string.

Python supports various different types of Literals. Let us look at each one of them in detail.

Numeric Literals

Numeric Literals are values assigned to the Variables or Constants which cannot be changed i.e., they are immutable. There are a total of 3 categories in Numeric Literals. They are – Integer, Float, and Complex.

Example

```
# Int Numeric Literal
a = 30

# Float Numeric Literal
b = 40.67

# Complex Numeric Literal
c = 10+4j

print(a)
print(b)
print(c)
print(c.real, c.imag)
```

Output

```
30
40.67
(10+4j)
10.0 4.0
```

To generate real and imaginary components of complex numbers, we utilize real literal (c.real) and imaginary literal (c.imag), respectively.

Long

Long literals were nothing but integers with unlimited length. From Python 2.2 and onwards, the integers that used to overflow were automatically converted into long ints. Since Python 3.0, the long literal has been dropped. What was the long data type in Python 2 is now the standard int type in Python 3.

Long literals used to be represented with a suffix- l or L. The usage of L was strongly recommended as l looked a lot like the digit 1.

Check out the following example to see how it was denoted-

Example

```
#usage long literal before it was depreciated
x=037467L
print(x)
```

Note– The code snippet was executed using Python 1.8. **Output**

```
Success #stdin

Success #stdin #stdout 0.01s 7320KB
16183
```

String Literals

A string literal is a series of characters surrounded by quotation marks. For a string, we can use single, double, or triple quotations. We can write multi-line strings or display them in the desired format by using triple quotes. A single character surrounded by single or double quotations is also known as a character literal.

Example

```
string = 'Hello Guys'
multi_line = '''Hey
   There!!'''
char = 'Z'

print(string)
print(multi_line)
print(char)
```

Output –

```

Hello Guys
Hey
   There!!
Z
```

Boolean Literals

A Boolean Literal has either of the 2 values – True or False. Where True is considered as 1 and False is considered as 0.

Example

```

boolean1 = (1 == True)
boolean2 = (1 == False)

num = 20
age = 20

x = True + 10
y = False + 50
```

```
print(boolean1)
print(boolean2)

print(num==age)

print('Value of x:', x)
print('Value of y:', y)
```

Output

```
True
False
True
Value of x: 11
Value of y: 50
```

True indicates a value of 1 in Python, while False represents a value of 0. Because 1 equals True, the value of boolean1 is True. And as 1 does not equal False, the value of boolean2 is False. Similarly, we can utilize True and False as values in numeric expressions.

Special Literals

Python provides a special kind of literal known as None. We use this type of Literal in order to specify the field has not been created. It also denotes the end of a list in Python.

Example

```
soap = "Available"
handwash = None

def items(x):
    if x == soap:
        print('Soap:', soap)
    else:
        print('Soap:', handwash)

items(soap)
items(handwash)
```

Output

```
Soap: Available
Soap: None
```

In the above program, we define a function named ‘item’. Inside the ‘item’ function, when we set the argument as ‘soap’ then, it displays ‘Available’. And, when the argument is ‘handwash’, it displays ‘None’.

Literal Collections

In Python, there are 4 different types of Literal Collections. They represent more complicated and complex data and assist Python scripts to be more extensible. Let us look at each one of them in detail.

1. List Literals

The elements in a list are of many data types. The values in the List are surrounded by square brackets ([]) and separated by commas (,). List values can be changed i.e., they are mutable.

Example –

```
cars = ['Tata', 'BMW', 'Audi', 'Ferrari']
student = ['John', 20, 9876432113, 'Mumbai']

print(cars)
print(student)
```

Output

```
['Tata', 'BMW', 'Audi', 'Ferrari']
['John', 20, 9876432113, 'Mumbai']
```

2. Tuple Literals

Just like a List, a tuple is also a collection of various data types. It is surrounded by parentheses ‘(),’ and each element is separated by a comma (,). It is unchangeable (immutable).

Example –

```
num = (1, 2, 4, 5, 7, 8)
student = ('John', 20, 9876432113, 'Mumbai')

print(num)
print(student)
```

Output –

```
(1, 2, 4, 5, 7, 8)
('John', 20, 9876432113, 'Mumbai')
```

3. Dict Literals

The data is stored in the dictionary as a key-value pair. It is surrounded by curly braces '{}', and each pair is separated by commas (,). A dictionary can hold various types of data. Dictionaries are subject to change.

Example –

```
student = {'Name':'David', 'Age':22, 'Sex':'Male', 'City':'California', }
print(student)

print(student.keys())
print(student.values()))
```

Output

```
{'Name': 'David', 'Age': 22, 'Sex': 'Male', 'City': 'California'}

dict_keys(['Name', 'Age', 'Sex', 'City'])
dict_values(['David', 22, 'Male', 'California'])
```

4. Set Literals

Set is an unordered data set collection. It is surrounded by and each element is separated by a comma (,).

Example

```
vowels = {'a', 'e', 'i', 'o', 'u'}
print(vowels)

chars = {'a', 'b', 'a', 'e', 'z', 'z', 'x'}
print(chars)  # Sets has no duplicate elements
```

Output

```
{'o', 'i', 'u', 'e', 'a'}
{'x', 'b', 'z', 'e', 'a'}
```

Constants in Python

In the Python programming language, Constants are types of variables whose values cannot be altered or changed after initialization. These values are universally proven to be true and they cannot be changed over time. Generally, python constants are declared and initialized on different modules/files.

To Create a constant in Python
The most important point we have to take into consideration while creating a constant in Python is that we need to create a separate python file for declaring constants and then importing this file in the main.py file. Let us understand python constants by looking at the below example.

Example

```
# constant.py file
PI = 3.14



# main.py file
import constant
print(‘Value of PI:’, constant.PI)
```

Output

```
Value of PI: 3.14
```

The basic difference between a variable, constant, and a literal in Python

Variables	Constants	Literals
A variable is a named location in memory where data is stored	A constant is a variable whose value cannot be modified.	Literals are raw values or data that are stored in a variable or constant.
Variables are mutable, i.e., their values can be changed and updated.	Constants are immutable, i.e. their values cannot be changed or updated.	Literals are both mutable or immutable depending on the type of literal used.

For example – num = 50 Here, num is the variable	For example – GRAVITY = 9.8 Here, GRAVITY is the constant	For example – str = ‘Hello’ Here, ‘Hello’ is literal. It is a string literal

The rules and conventions to be followed while naming a variable or constant?
Python programming language incorporates certain rules which should be followed while declaring a Variable or a Constant in the program. They are listed as follows –

1. Constant and variable names should contain a combination of lowercase (a-z) or capital (A-Z) characters, numbers (0-9), or an underscore ( ).
2. If you want to use underscore to separate two words in a variable name, do so.
3. To declare a constant, use all capital letters as feasible.
4. Never use special characters such as !, @, #, $, %, and so on.
5. A variable/constant name should never begin with a number.

Keywords in Python

Python Keywords are some predefined and reserved words in python that have special meanings. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, or variable name. All the keywords in python are written in lowercase except True and False. There are 35 keywords in Python 3.11.

In python, there is an inbuilt keyword module that provides an iskeyword() function that can be used to check whether a given string is a valid keyword or not. Furthermore we can check the name of the keywords in Python by using the kwlist attribute of the keyword module.

Rules for Keywords in Python

- Python keywords cannot be used as identifiers.
- All the keywords in python should be in lowercase except True and False.

List of Python Keywords

Keywords	**Description**
and	This is a logical operator which returns true if both the operands are true else returns false.
or	This is also a logical operator which returns true if anyone operand is true else returns false.

Keywords	Description
not	This is again a logical operator it returns True if the operand is false else returns false.
if	This is used to make a conditional statement.
elif	Elif is a condition statement used with an if statement. The elif statement is executed if the previous conditions were not true.
else	Else is used with if and elif conditional statements. The else block is executed if the given condition is not true.
for	This is used to create a loop.
while	This keyword is used to create a while loop.
break	This is used to terminate the loop.
as	This is used to create an alternative.
def	It helps us to define functions.
lambda	It is used to define the anonymous function.
pass	This is a null statement which means it will do nothing.
return	It will return a value and exit the function.
True	This is a boolean value.
False	This is also a boolean value.
try	It makes a try-except statement.
with	The with keyword is used to simplify exception handling.
assert	This function is used for debugging purposes. Usually used to check the correctness of code

Keywords	**Description**
class	It helps us to define a class.
continue	It continues to the next iteration of a loop
del	It deletes a reference to an object.
except	Used with exceptions, what to do when an exception occurs
finally	Finally is used with exceptions, a block of code that will be executed no matter if there is an exception or not.
from	It is used to import specific parts of any module.
global	This declares a global variable.
import	This is used to import a module.
in	It's used to check whether a value is present in a list, range, tuple, etc.
is	This is used to check if the two variables are equal or not.
none	This is a special constant used to denote a null value or avoid. It's important to remember, 0, any empty container(e.g empty list) do not compute to None
nonlocal	It's declared a non-local variable.
raise	This raises an exception.
yield	It ends a function and returns a generator.
async	It is used to create asynchronous coroutine.
await	It releases the flow of control back to the event loop.

The following code allows you to view the complete list of Python's keywords.

This code imports the "keyword" module in Python and then prints a list of all the keywords in Python using the "kwlist" attribute of the "keyword" module. The "kwlist" attribute is a list of strings, where each string represents a keyword in Python. By printing this list, we can see all the keywords that are reserved in Python and cannot be used as identifiers.

```
# code
import keyword
print(keyword.kwlist)
```

Output

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

Identifiers in Python

Identifier is a user-defined name given to a variable, function, class, module, etc. The identifier is a combination of character digits and an underscore. They are case-sensitive i.e., 'num' and 'Num' and 'NUM' are three different identifiers in python. It is a good programming practice to give meaningful names to identifiers to make the code understandable.

We can also use the Python string isidentifier() method to check whether a string is a valid identifier or not.

Rules for Naming Python Identifiers

- It cannot be a reserved python keyword.
- It should not contain white space.
- It can be a combination of A-Z, a-z, 0-9, or underscore.
- It should start with an alphabet character or an underscore (_).
- It should not contain any special character other than an underscore (_).

Examples of Python Identifiers

Valid identifiers:

- var1
- _var1
- _1_var
- var_1

Invalid Identifiers

- !var1
- 1var
- 1_var
- var#1
- var 1

Python Keywords and Identifiers Examples

Example 1: Example of and, or, not, True, False keywords.

```
print("example of True, False, and, or, not keywords")
```

```
# compare two operands using and operator
print(True and True)

# compare two operands using or operator
print(True or False)

# use of not operator
print(not False)
```

Output:

```
example of True, False, and, or, not keywords

True

True

True
```

Example 2: Example of a break, continue keywords and identifier.

```
# execute for loop
for i in range(1, 11):

        # print the value of i
        print(i)

        # check the value of i is less than 5
        # if i lessthan 5 then continue loop
        if i < 5:
                continue

        # if i greater than 5 then break loop
        else:
                break
```

Output:

In this example, ‘if’ and ‘else’ keywords are used to specify some conditions. Whereas ‘i’ is a user-defined variable which is an identifier that temporarily stores the value from 1 to 10.

```
1

2

3

4

5
```

Example 3: example of for, in, if, elif, and else keywords.

```
# run for loop
for t in range(1, 5):
# print one of t ==1
        if t == 1:
                print('One')
```

```
# print two if t ==2
        elif t == 2:
                print('Two')
        else:
                print('else block execute')
```
Output:

```
One

Two

else block execute

else block execute
```

Example 4: Example of def, if, and else keywords.

```
# define GFG() function using def keyword
def GFG():
        i=20
        # check i is odd or not
        # using if and else keyword
        if(i % 2 == 0):
                print("given number is even")
        else:
                print("given number is odd")

# call GFG() function
GFG()
```
Output:

In this example, 'def' keyword is used to define the GFG() function. Whereas 'i' and 'GFG' are identifiers.

```
given number is even
```

Example 7: use of return keyword.

```
# define a function
def fun():
# declare a variable
        a = 5
        # return the value of a
        return a
# call fun method and store
# it's return value in a variable
t = fun()
# print the value of t
print(t)
```
Output: 5

Keywords and Identifiers in Python

Keywords in Python are predefined words that have a special meaning to the interpreter. They are reserved words that are used to perform a specific task in Python programming.
Identifiers in Python are names given to different parts of a Python program like variables, functions, classes, etc. They are user-defined and the users must follow a set of rules to define them in a python program.

Difference between a Keyword and an Identifier in Python

A Keyword in Python is a predefined reserved word that is meaningful to the interpreter and performs a specific task. Whereas, an identifier is a user-defined word given to different parts of python programming. An identifier can be a name given to a variable, function or a class.

Python Operators

Operators are special symbols that perform operations on variables and values. Operators allow us to perform various computations, comparisons, and manipulations on data in a programming language. They are a fundamental part of any programming language and play a crucial role in writing expressions and statements to achieve specific tasks.

Operators can be categorized into different types based on their functionality:

For example,

```
print(5 + 6) # 11
```
Run Code

Here, `+` is an operator that adds two numbers: **5** and **6**.
Types of Python Operators

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

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

1. Python Arithmetic Operators

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

```
sub = 10 - 5 # 5
```

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

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)
```

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
- - t subtract b from a
- * multiply a and b
- / t 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
var 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

```
# assign 10 to a
a = 10

# assign 5 to b
b = 5

# assign the sum of a and b to a
a += b      # a = a + b

print(a)

# Output: 15
```

Here, we have used the += operator to assign the sum of a and b to a.
Similarly, we can use any other assignment operators according to the need.

3. Python Comparison Operators

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

```
a = 5
b =2

print (a > b)    # True
```

Run Code

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

# equal to operator
print('a == b =', a == b)

# not equal to operator
print('a != b =', a != b)

# greater than operator
print('a > b =', a > b)

# less than operator
print('a < b =', a < b)

# greater than or equal to operator
print('a >= b =', a >= b)

# less than or equal to operator
print('a <= b =', a <= b)
```

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. 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))    # 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

```
# logical AND
print(True and True)     # True
print(True and False)    # False

# logical OR
print(True or False)     # True


# logical NOT
print(not True)          # False
```

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

5. Python Bitwise operators

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

For example, **2** is `10` in binary and **7** is `111`.
In the table below: Let `x` = 10 (`0000 1010` in binary) and `y` = 4 (`0000 0100` in binary)

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 << 2 = 40 (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 on the same part of the memory. Two variables that are equal does not imply that 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)  # prints False

print(x2 is y2) # prints True

print(x3 is y3)  # prints False
```

Here, we see that x1 and y1 are integers of the same values, so they are equal as well as identical. 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, list, tuple, set and dictionary).
In a dictionary we can only test for presence of 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

```
x = 'Hello world'
y = {1:'a', 2:'b'}

# check if 'H' is present in x string
print('H' in x) # prints True

# check if 'hello' is present in x string
print('hello' not in x) # prints True

# check if '1' key is present in y
print(1 in y) # prints True

# check if 'a' key is present in y
print('a' in y)  # prints False
```

Run Code

Output

```
True
True
True
False
```

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similarly, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

Python - Data Types

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

Every value has a datatype, and variables can hold values. Python is a powerfully composed language; consequently, we don't have to characterize the sort of variable while announcing it. The interpreter binds the value implicitly to its type.

```
a = 5
```

We did not specify the type of the variable a, which has the value five from an integer. The Python interpreter will automatically interpret the variable as an integer.

We can verify the type of the program-used variable thanks to Python. The type() function in Python returns the type of the passed variable.

Consider the following illustration when defining and verifying the values of various data types.

```
1. a=10
2. b="Hi Python"
3. c = 10.5
4. print(type(a))
5. print(type(b))
6. print(type(c))
```

Output:

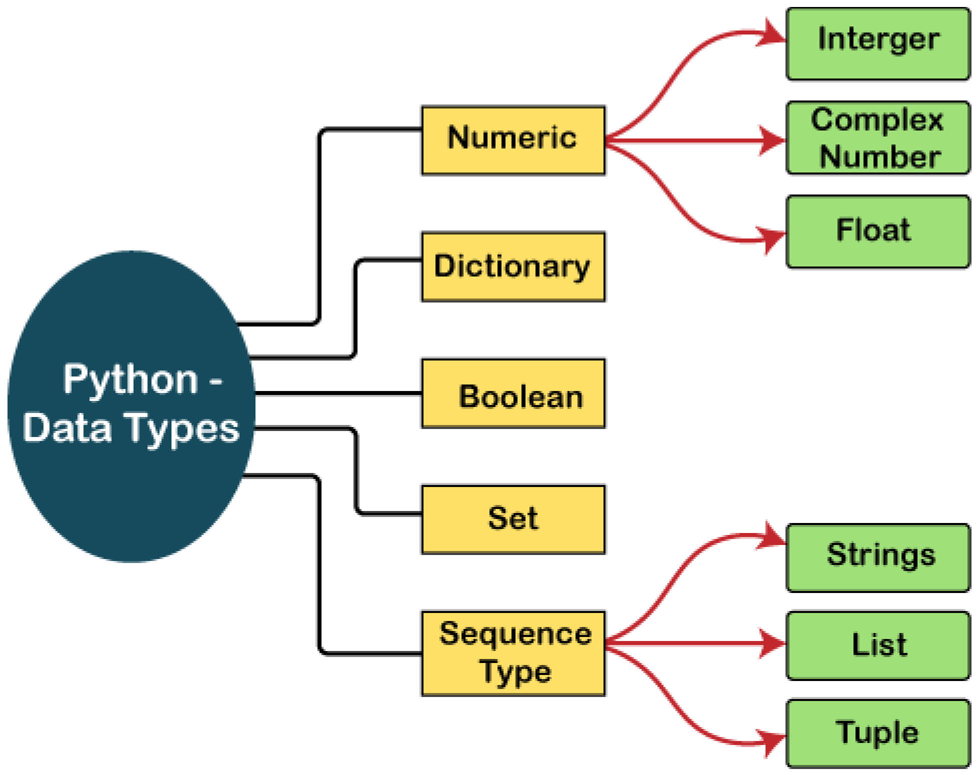
```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can contain a variety of values. On the other hand, a person's id must be stored as an integer, while their name must be stored as a string.

The storage method for each of the standard data types that Python provides is specified by Python. The following is a list of the Python-defined data types.

1. Numbers
2. Sequence Type
3. Boolean
4. Set
5. Dictionary



The data types will be briefly discussed in this tutorial section. We will talk about every single one of them exhaustively later in this instructional exercise.

Numbers

Numeric values are stored in numbers. The whole number, float, and complex qualities have a place with a Python Numbers datatype. Python offers the type() function to determine a variable's data type. The instance () capability is utilized to check whether an item has a place with a specific class.

When a number is assigned to a variable, Python generates Number objects. For instance,

```
a = 5
print("The type of a", type(a))

b = 40.5
print("The type of b", type(b))

c = 1+3j
print("The type of c", type(c))
print(" c is a complex number", isinstance(1+3j,complex))
```

Output:

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
c is complex number: True
```

Python supports three kinds of numerical data.

- **Int:** Whole number worth can be any length, like numbers 10, 2, 29, - 20, - 150, and so on. An integer can be any length you want in Python. Its worth has a place with int.
- **Float:** Float stores drifting point numbers like 1.9, 9.902, 15.2, etc. It can be accurate to within 15 decimal places.
- **Complex:** An intricate number contains an arranged pair, i.e., x + iy, where x and y signify the genuine and non-existent parts separately. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Sequence Type

String

The sequence of characters in the quotation marks can be used to describe the string. A string can be defined in Python using single, double, or triple quotes.

String dealing with Python is a direct undertaking since Python gives worked-in capabilities and administrators to perform tasks in the string.

When dealing with strings, the operation "hello"+" python" returns "hello python," and the operator + is used to combine two strings.

Because the operation "Python" *2 returns "Python," the operator * is referred to as a repetition operator.

The Python string is demonstrated in the following example.

Example - 1

```
str = "string using double quotes"
print(str)
s = '''A multiline
string'''
print(s)
```

Output:

```
string using double quotes
A multiline
string
```

Look at the following illustration of string handling.

Example - 2

```
str1 = 'hello javatpoint' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

```
he
o
hello javatpointhello javatpoint
hello javatpoint how are you
```

List

Lists in Python are like arrays in C, but lists can contain data of different types. The things put away in the rundown are isolated with a comma (,) and encased inside square sections [].

To gain access to the list's data, we can use slice [:] operators. Like how they worked with strings, the list is handled by the concatenation operator (+) and the repetition operator (*).

Look at the following example.

Example:

```
list1  = [1, "hi", "Python", 2]
#Checking type of given list
print(type(list1))

#Printing the list1
print (list1)

# List slicing
print (list1[3:])

# List slicing
print (list1[0:2])

# List Concatenation using + operator
print (list1 + list1)

# List repetation using * operator
```

```
print (list1 * 3)
```

Output:

```
[1, 'hi', 'Python', 2]
[2]
[1, 'hi']
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
```

Tuple

In many ways, a tuple is like a list. Tuples, like lists, also contain a collection of items from various data types. A parenthetical space () separates the tuple's components from one another.

Because we cannot alter the size or value of the items in a tuple, it is a read-only data structure.

Let's look at a straightforward tuple in action.

Example:

```
tup  = ("hi", "Python", 2)
# Checking type of tup
print (type(tup))

#Printing the tuple
print (tup)

# Tuple slicing
print (tup[1:])
print (tup[0:1])

# Tuple concatenation using + operator
print (tup + tup)

# Tuple repatation using * operator
print (tup * 3)

# Adding value to tup. It will throw an error.
t[2] = "hi"
```

Output:

```
<class 'tuple'>
('hi', 'Python', 2)
('Python', 2)
('hi',)
```

```
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)

Traceback (most recent call last):
  File "main.py", line 14, in <module>
    t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary

A dictionary is a key-value pair set arranged in any order. It stores a specific value for each key, like an associative array or a hash table. Value is any Python object, while the key can hold any primitive data type.

The comma (,) and the curly braces are used to separate the items in the dictionary.

Look at the following example.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}

 # Printing dictionary
 print (d)

 # Accesing value using keys
 print("1st name is "+d[1])
 print("2nd name is "+ d[4])

print (d.keys())
print (d.values())
```

Output:

```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
dict_keys([1, 2, 3, 4])
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

Boolean

True and False are the two default values for the Boolean type. These qualities are utilized to decide the given assertion valid or misleading. The class book indicates this. False can be represented by the 0 or the letter "F," while true can be represented by any value that is not zero.

Look at the following example.

```
# Python program to check the boolean type
```

```
print(type(True))
print(type(False))
print(false)
```

Output:

```
<class 'bool'>
<class 'bool'>
NameError: name 'false' is not defined
```

Set

The data type's unordered collection is Python Set. It is iterable, mutable(can change after creation), and has remarkable components. The elements of a set have no set order; It might return the element's altered sequence. Either a sequence of elements is passed through the curly braces and separated by a comma to create the set or the built-in function set() is used to create the set. It can contain different kinds of values.

Look at the following example.

```
# Creating Empty set
set1 = set()

set2 = {'James', 2, 3,'Python'}

#Printing Set value
print(set2)

# Adding element to the set

set2.add(10)
print(set2)

#Removing element from the set
set2.remove(2)
print(set2)
```

Output:

```
{3, 'Python', 'James', 2}
{'Python', 'James', 3, 2, 10}
{'Python', 'James', 3, 10}
```

Python Input Statement:

In Python, we use **the input()** function to take input from the user. Whatever you enter as input, the input function converts it into a string. If you enter an integer value still input() function converts it into a string.

Example:

- Python3

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

Output :

```
What is your name? GFG

Hello, GFG!
```

Python input() Syntax

Syntax: *input(prompt)*

Parameter:

- ***Prompt:*** *(optional) The string that is written to standard output(usually screen) without newline.*

Return: *String object*

input() in Python Examples

Taking input in Python

In this example, we are using the Python input() function which takes input from the user and prints it.

```
aking input from the user

string = input()


# Output

print(string)
```

Output:

```
geeksforgeeks
```

User Input in Python

In this example, we are taking input from the user with a prompt and printing it.

```
# Taking input from the user

name = input("Enter your name")



# Output

print("Hello", name)
```

Output:

```
Enter your name:ankit rai

Hello ankit rai
```

Convert User Input to a Number

In this example, we are using the Python input() function which takes input from the user in string format converting it into an integer adding 1 to the integer, and printing it.

- Python3

```
# Taking input from the user as integer

num = int(input("Enter a number:"))

add = num + 1



# Output

print(add)
```

Output:

```
Enter a number:15

16
```

Take float input in Python

In this example, we are using the Python input() function which takes input from the user in string format converts it into float adds 1 to the float, and prints it.

- Python3

```
# Taking input from the user as float


num =float(input("Enter number "))

add = num + 1


# output

print(add)
```

Output:

```
Enter number 5

6.0
```

Python Accept List as a input From User

In this example, we are taking input from the user in string format converting it into a list, and printing it.

- Python3

```
# Taking input from the user as list


li =list(input("Enter number "))


# output

print(li)
```

Output:

```
Enter number 12345

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

Take User Input for Tuples and Sets

In this example, we are taking input from the user in string format converting it into a tuple, and printing it.

- Python3

```
# Taking input from the user as tuple


num =tuple(input("Enter number "))


# output

print(num)
```

Output:

```
Enter number 123

('1', '2', '3')
```

Using input with a dictionary comprehension

In this example, we are taking the words separated by space and we make a dictionary of the word as key with their length as value.

- Python3

```
words_str = input("Enter a list of words, separated by spaces: ")

words = {word: len(word) for word in words_str.split()}

print(words)
```

Output :

```
Enter a list of words, separated by spaces: geeks for geeks

{'geeks': 5, 'for': 3}
```

Output Statement in Python:

Python print() function prints the message to the screen or any other standard output device.
Example
In this example, we have created three variables integer, string and float and we are printing all the variables with print() function in Python.

- Python3

```
name = "John"

age = 30


print("Name:", name)

print("Age:", age)
```

Output
```
Name: John
Age: 30
```
Python print() Function Syntax
Syntax : *print(value(s), sep= ‘ ‘, end = ‘\n’, file=file, flush=flush)*
Parameters:

- ***value(s):*** *Any value, and as many as you like. Will be converted to a string before printed*
- ***sep=’separator’ :*** *(Optional) Specify how to separate the objects, if there is more than one.Default :’ ‘*
- ***end=’end’:*** *(Optional) Specify what to print at the end.Default : ‘\n’*
- ***file :*** *(Optional) An object with a write method. Default :sys.stdout*
- ***flush :*** *(Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False*

Return Type: *It returns output to the screen.*
Though it is not necessary to pass arguments in the print() function, it requires an empty parenthesis at the end that tells Python to execute the function rather than calling it by name. Now, let’s explore the optional arguments that can be used with the print() function.

How print() works in Python?
You can pass variables, strings, numbers, or other data types as one or more parameters when using the print() function. Then, these parameters are represented as strings by their respective str() functions. To create a single output string, the transformed strings are concatenated with spaces between them.
In this code, we are passing two parameters name and age to the print function.

- Python3

```
name = "Alice"
age = 25


print("Hello, my name is", name, "and I am", age, "years old.")
```

Output :

```
Hello, my name is Alice and I am 25 years old.
```

Python print() Function with Examples

Python String Literals

String literals in Python's print statement are primarily used to format or design how a specific string appears when printed using the print() function.

- **\n:** This string literal is used to add a new blank line while printing a statement.
- **"":** An empty quote ("") is used to print an empty line.

Example

This code uses \n to print the data to the new line.

- Python3

```
print("GeeksforGeeks \n is best for DSA Content.")
```

Output

```
GeeksforGeeks
 is best for DSA Content.
```

Python "end" parameter in print()

The end keyword is used to specify the content that is to be printed at the end of the execution of the print() function. By default, it is set to "\n", which leads to the change of line after the execution of print() statement.

Example

In this example, we are using print() with end and without end parameters.

- Python3

```
# This line will automatically add a new line before the
# next print statement
print ("GeeksForGeeks is the best platform for DSA content")


# This print() function ends with "**" as set in the end argument.
print ("GeeksForGeeks is the best platform for DSA content", end= "**")
```

```
print("Welcome to GFG")
```

Output

```
GeeksForGeeks is the best platform for DSA content
GeeksForGeeks is the best platform for DSA content**Welcome to GFG
```

flush parameter in Python with print() function

The I/Os in Python are generally buffered, meaning they are used in chunks. This is where flush comes in as it helps users to decide if they need the written content to be buffered or not. By default, it is set to false. If it is set to true, the output will be written as a sequence of characters one after the other. This process is slow simply because it is easier to write in chunks rather than writing one character at a time. To understand the use case of the flush argument in the print() function, let’s take an example.

Example

Imagine you are building a countdown timer, which appends the remaining time to the same line every second. It would look something like below:

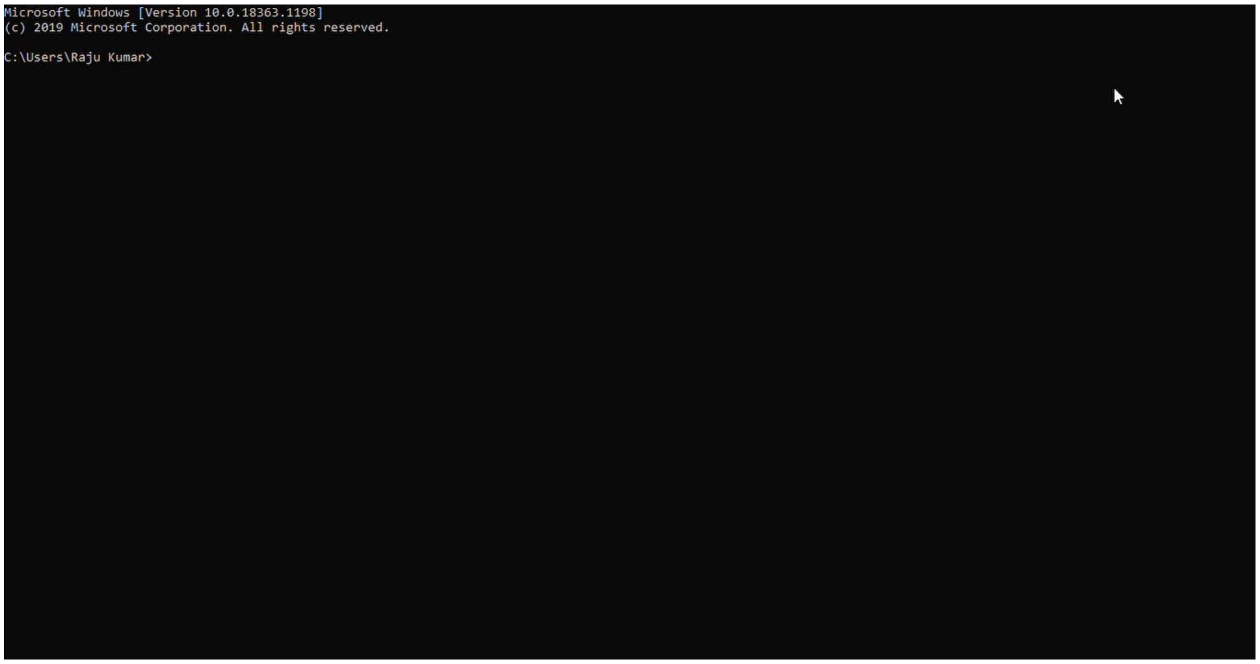
```
3>>>2>>>1>>>Start
```

The initial code for this would look something like below as follows:

- Python3

```
import time



count_seconds = 3

for i in reversed(range(count_seconds + 1)):

    if i > 0:

        print(i, end='>>>')

        time.sleep(1)

    else:

        print('Start')
```

So, the above code adds text without a trailing newline and then sleeps for one second after each text addition. At the end of the countdown, it prints Start and terminates the line. If you run the code as it is, it waits for 3 seconds and abruptly prints the entire text at once. This is a waste of 3 seconds caused due to buffering of the text chunk as shown below :



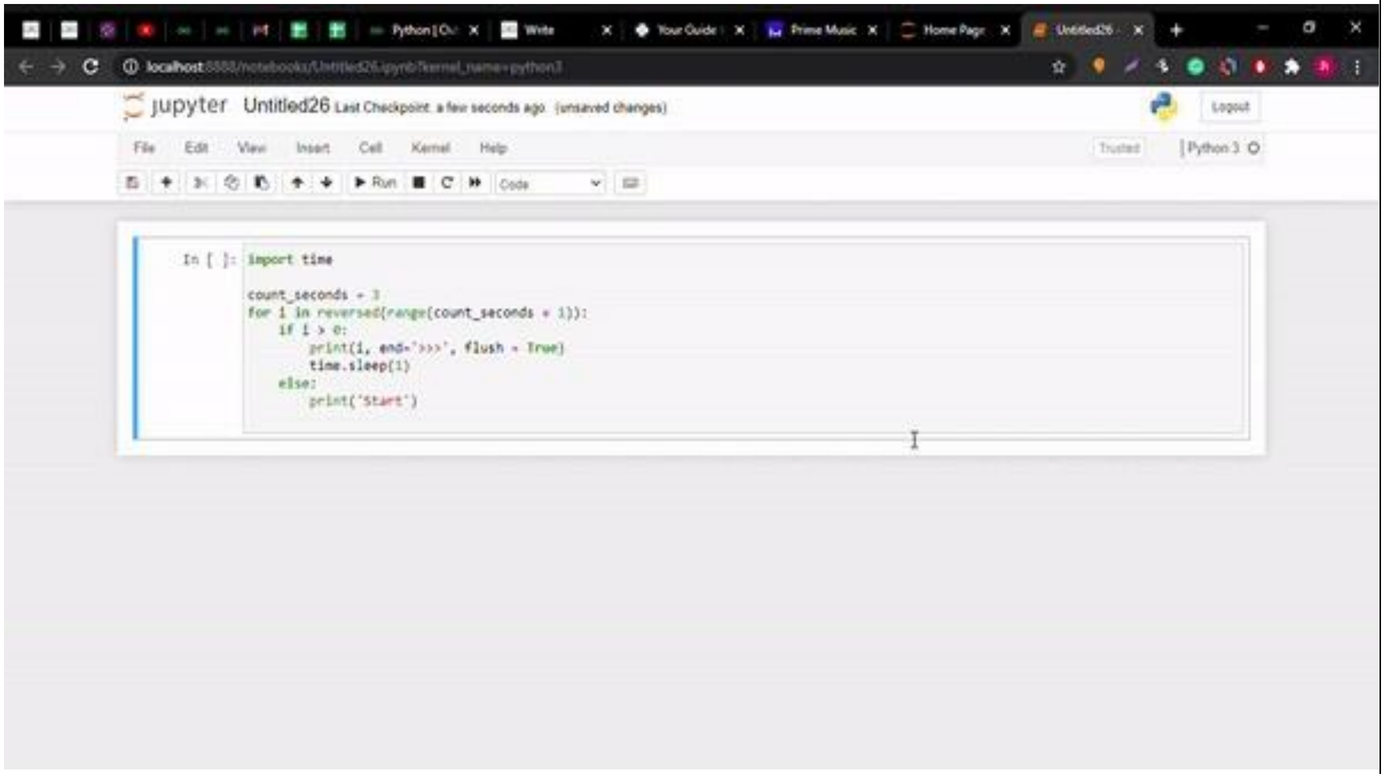
```
Microsoft Windows [Version 10.0.18363.1198]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Raju Kumar>
```

Though buffering serves a purpose, it can result in undesired effects as shown above. To counter the same issue, the flush argument is used with the print() function. Now, set the flush argument as true and again see the results.

- Python3

```
import time


count_seconds = 3
for i in reversed(range(count_seconds + 1)):
    if i > 0:
        print(i, end='>>>', flush = True)
        time.sleep(1)
    else:
        print('Start')
```

Output



```
In [ ]: import time

        count_seconds = 3
        for i in reversed(range(count_seconds + 1)):
            if i > 0:
                print(i, end='>>>', flush = True)
                time.sleep(1)
            else:
                print('Start')
```

Python print() flush argument

Python "sep" parameter in print()
The print() function can accept any number of positional arguments. To separate these positional arguments, the keyword argument "sep" is used.
Note: As sep, end, flush, and file are keyword arguments their position does not change the result of the code.
Example
This code is showing that how can we use the sep argument for multiple variables.

- Python3

```
a=12

b=12

c=2022

print(a,b,c,sep="-")
```

Output :
12-12-2022
Example
Positional arguments cannot appear after keyword arguments. In the below example **10**, **20** and **30** are positional arguments where **sep=' – '** is a keyword argument.

- Python3

```
print(10, 20, sep=' - ', 30)
```

Output :

```
  File "0b97e8c5-bacf-4e89-9ea3-c5510b916cdb.py", line 1
    print(10, 20, sep=' - ', 30)
                          ^
SyntaxError: positional argument follows keyword argument
```

File Argument in Python print()

Contrary to popular belief, the print() function doesn’t convert messages into text on the screen. These are done by lower-level layers of code, that can read data(message) in bytes. The print() function is an interface over these layers, that delegates the actual printing to a stream or **file-like object**. By default, the print() function is bound to **sys.stdout** through the file argument.

With IO Module

This code creates a dummy file using the io module in Python. It then adds a message “**Hello Geeks!!**” to the file using the print() function and specifies the file parameter as the dummy file.

- Python3

```
import io


# declare a dummy file
dummy_file = io.StringIO()


# add message to the dummy file
print('Hello Geeks!!', file=dummy_file)


# get the value from dummy file
print(dummy_file.getvalue())
```

Output

```
Hello Geeks!!
```

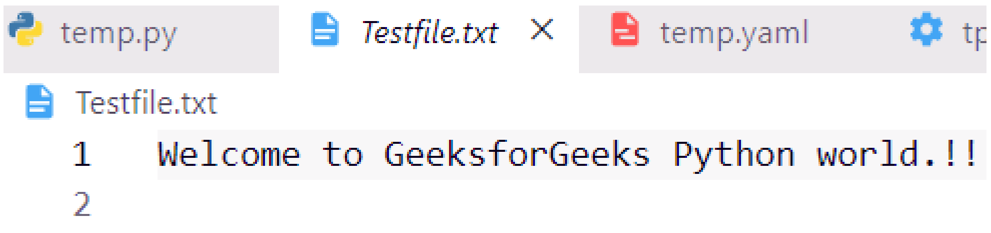
Writing to a File with Python’s print() Function

This code is writing the data in the **print()** function to the text file.

- Python3

```
print('Welcome to GeeksforGeeks Python world.!!', file=open('Testfile.txt', 'w'))
```

Output



```
Welcome to GeeksforGeeks Python world.!!
```

There are several ways to deliver the output of a program. In Python, we use the print() function to output data to the screen. Sometimes we might want to take input from the user. We can do so by using the input() function. Python takes all the input as a string input by default. To convert it to any other data type, we have to convert the input explicitly.

Python Comments

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program. Comments enhance the readability of the code and help the programmers to understand the code very carefully.
There are three types of comments in Python:

- Single line Comments
- Multiline Comments
- Docstring Comments

Python3

```
# sample comment

name = "geeksforgeeks"

print(name)
```

Output:
geeksforgeeks

Types of Comments in Python

Single-Line Comments in Python
Python single-line comment starts with the hashtag symbol (#) with no white spaces and lasts till the end of the line. If the comment exceeds one line then put a hashtag on the next line and continue the Python Comments. Python's single-line comments are proved useful for supplying short explanations for variables, function declarations, and expressions. See the following code snippet demonstrating single line comment:

Example:
Python3

```
# Print “GeeksforGeeks !” to console

print("GeeksforGeeks")
```

Output

```
GeeksforGeeks
```

Multi-Line Comments in Python

Python does not provide the option for multiline comments. However, there are different ways through which we can write multiline comments.
Multiline comments using multiple hashtags (#)
We can multiple hashtags (#) to write multiline comments in Python. Each and every line will be considered as a single-line comment.

Python3

```
# Python program to demonstrate

# multiline comments

print("Multiline comments")
```

Output

```
Multiline comments
```

String Literals In Python

Python ignores the string literals that are not assigned to a variable so we can use these string literals as Python Comments**.**
Single-line comments using string literals
On executing the above code we can see that there will not be any output so we use the strings with triple quotes(“””) as multiline comments.
Python3

```
'This will be ignored by Python'
```

Multiline comments using string literals
Python3

```
""" Python program to demonstrate
```

```
 multiline comments"""

print("Multiline comments")
```

Output

```
Multiline comments
```

Docstring in Python

Python docstring is the string literals with triple quotes that are appeared right after the function. It is used to associate documentation that has been written with Python modules, functions, classes, and methods. It is added right below the functions, modules, or classes to describe what they do. In Python, the docstring is then made available via the __doc__ attribute.

Example:

Python3

```
def multiply(a, b):

    """Multiplies the value of a and b"""

    return a*b




# Print the docstring of multiply function

print(multiply._doc_)
```

Output:

```
Multiplies the value of a and b
```

Advantages of comments in Python

Comments are generally used for the following purposes:

- Code Readability
- Explanation of the code or Metadata of the project
- Prevent execution of code
- To include resources

Expression in Python

A combination of operands and operators is called an **expression**. The expression in Python produces some value or result after being interpreted by the Python interpreter. An expression in Python is a combination of operators and operands.

An example of expression can be : ��=��+10x=x+10. In this expression, the first 1010 is added to the variable x. After the addition is performed, the result is assigned to the variable x.

Example :

```
x = 25          # a statement
x = x + 10      # an expression

print(x)
```

Output :

```
35
```

An expression in Python is very different from statements in Python. A statement is not evaluated for some results. A statement is used for creating variables or for displaying values.

Example :

```
a = 25      # a statement

print(a)    # a statement
```

Output :

```
25
```

An expression in Python can contain **identifiers**, **operators**, and **operands**. Let us briefly discuss them.

An **identifier** is a name that is used to define and identify a class, variable, or function in Python.

An **operand** is an object that is operated on. On the other hand, an **operator** is a special symbol that performs the arithmetic or logical computations on the operands. There are many types of operators in Python, some of them are :

- **+ :** add (plus).
- **- :** subtract (minus).
- **x :** multiply.
- **/ :** divide.

- **** :** power.
- **% :** modulo.
- << **:** left shift.
- >> **:** right shift.
- **& :** bit-wise AND.
- **| :** bit-wise OR.
- ^ **:** bit-wise XOR.
- ~ **:** bit-wise invert.
- < **:** less than.
- > **:** greater than.
- <= **:** less than or equal to.
- >= **:** greater than or equal to.
- == **:** equal to.
- **!= :** not equal to.
- **and :** boolean AND.
- **or :** boolean OR.
- **not :** boolean NOT.

The **expression** in Python can be considered as a logical line of code that is evaluated to obtain some result. If there are various operators in an expression then the operators are resolved based on their precedence. We have various types of expression in Python, refer to the next section for a more detailed explanation of the types of expression in Python.

Types of Expression in Python

We have various types of expression in Python, let us discuss them along with their respective examples.

1. Constant Expressions

A constant expression in Python that contains only constant values is known as a constant expression. In a **constant expression** in Python, the operator(s) is a constant. A constant is a value that cannot be changed after its initialization.

Example :

```
x = 10 + 15

# Here both 10 and 15 are constants but x is a variable.
print("The value of x is: ", x)
```

Output :

```
The value of x is: 25
```

2. Arithmetic Expressions

An expression in Python that contains a combination of operators, operands, and sometimes parenthesis is known as an **arithmetic expression**. The result of an arithmetic expression is also a numeric value just like the constant expression discussed above. Before getting into the example of an arithmetic expression in Python, let us first know about the various operators used in the arithmetic expressions.

Operator	**Syntax**	**Working**
+	x + y	Addition or summation of x and y.
-	x - y	Subtraction of y from x.
x	x x y	Multiplication or product of x and y.
/	x / y	Division of x and y.
//	x // y	Quotient when x is divided by y.
%	x % y	Remainder when x is divided by y.
**	x ** y	Exponent (x to the power of y).

Example :

```
x = 10
y = 5

addition = x + y
subtraction = x - y
product = x * y
division = x / y
power = x**y

print("The sum of x and y is: ", addition)
print("The difference between x and y is: ", subtraction)
print("The product of x and y is: ", product)
print("The division of x and y is: ", division)
print("x to the power y is: ", power)
```

Output :

```
The sum of x and y is: 15
The difference between x and y is:  5
The product of x and y is: 50
The division of x and y is: 2.0
```

```
x to the power y is: 100000
```

3. Integral Expressions

An **integral expression** in Python is used for computations and type conversion (**integer** to **float**, a **string** to **integer**, etc.). An integral expression always produces an integer value as a resultant.

Example :

```
x = 10      # an integer number
y = 5.0     # a floating point number

# we need to convert the floating-point number into an integer or vice versa for summation.
result = x + int(y)

print("The sum of x and y is: ", result)
```

Output :

```
The sum of x and y is: 15
```

4. Floating Expressions

A **floating expression** in Python is used for computations and type conversion (**integer** to **float**, a **string** to **integer**, etc.). A floating expression always produces a floating-point number as a resultant.

Example :

```
x = 10      # an integer number
y = 5.0     # a floating point number

# we need to convert the integer number into a floating-point number or vice versa for
summation.
result = float(x) + y

print("The sum of x and y is: ", result)
```

Output :

```
The sum of x and y is: 15.0
```

5. Relational Expressions

A **relational expression** in Python can be considered as a combination of two or more arithmetic expressions joined using relational operators. The overall expression results in either True or False (boolean result). We have four types of relational operators in Python

(i.e. >,<,>=,<=)(*i.e.*>,<,>=,<=).

A relational operator produces a boolean result so they are also known as **Boolean Expressions**.

For example :
10+15>2010+15>20

In this example, first, the arithmetic expressions (i.e. 10+1510+15 and 2020) are evaluated, and then the results are used for further comparison.

Example :

```
a = 25
b = 14
c = 48
d = 45

# The expression checks if the sum of (a and b) is the same as the difference of (c and d).
result = (a + b) == (c - d)
print("Type:", type(result))
print("The result of the expression is: ", result)
```

Output :

```
Type: <class 'bool'>
The result of the expression is:  False
```

6. Logical Expressions

As the name suggests, a logical expression performs the logical computation, and the overall expression results in either True or False (boolean result). We have three types of logical expressions in Python, let us discuss them briefly.

Operator	Syntax	Working
and	*x* and *y*	The expression return True if both *x* and *y* are true, else it returns False.
or	*x* or *y*	The expression return True if at least one of *x* or *y* is True.

Operator	Syntax	Working
not	not *x*	The expression returns True if the condition of *x* is False.

Note :
In the table specified above, *x* and *y* can be values or another expression as well.

Example :

```
from operator import and_


x = (10 == 9)
y = (7 > 5)


and_result = x and y
or_result = x or y
not_x = not x

print("The result of x and y is: ", and_result)
print("The result of x or y is: ", or_result)
print("The not of x is: ", not_x)
```

Output :

```
The result of x and y is: False
The result of x or y is: True
The not of x is: True
```

7. Bitwise Expressions

The expression in which the operation or computation is performed at the bit level is known as a **bitwise expression** in Python. The bitwise expression contains the bitwise operators.

Example :

```
x = 25
left_shift = x << 1
right_shift = x >> 1

print("One right shift of x results: ", right_shift)
print("One left shift of x results: ", left_shift)
```

Output :

```
One right shift of x results: 12
One left shift of x results: 50
```

8. Combinational Expressions

As the name suggests, a **combination expression** can contain a single or multiple expressions which result in an integer or boolean value depending upon the expressions involved.

Example :

```
x = 25
y = 35

result = x + (y << 1)

print("Result obtained : ", result)
```

Output :

```
Result obtained: 95
```

Whenever there are multiple expressions involved then the expressions are resolved based on their precedence or priority. Let us learn about the precedence of various operators in the following section.

Multiple Operators in Expression (Operator Precedence) ?

The operator precedence is used to define the operator's priority i.e. which operator will be executed first. The operator precedence is similar to the BODMAS rule that we learned in mathematics. Refer to the list specified below for operator precedence.

Precedence	Operator	Name
1.	( ) [ ] { }	Parenthesis
2.	**	Exponentiation
3.	-value , +value , ~value	Unary plus or minus, complement
4.	/ * // %	Multiply, Divide, Modulo
5.	+ –	Addition & Subtraction
6.	>> <<	Shift Operators

Precedence	Operator	Name
7.	&	Bitwise AND
8.	^	Bitwise XOR
9.	pipe symbol	Bitwise OR
10.	>= <= > <	Comparison Operators
11.	== !=	Equality Operators
12.	= += -= /= *=	Assignment Operators
13.	is, is not, in, not in	Identity and membership operators
14.	and, or, not	Logical Operators

Let us take an example to understand the precedence better :

```
x = 12
y = 14
z = 16

result_1 = x + y * z
print("Result of 'x + y + z' is: ", result_1)

result_2 = (x + y) * z
print("Result of '(x + y) * z' is: ", result_2)

result_3 = x + (y * z)
print("Result of 'x + (y * z)' is: ", result_3)
```

Output :

```
Result of 'x + y + z' is: 236
Result of '(x + y) * z' is: 416
Result of 'z + (y * z)' is: 236
```

Difference between Statements and Expressions in Python

We have earlier discussed statement expression in Python, let us learn the differences between them.

Statement in Python	Expression in Python

Statement in Python	**Expression in Python**
A statement in Python is used for creating variables or for displaying values.	The expression in Python produces some value or result after being interpreted by the Python interpreter.
A statement in Python is not evaluated for some results.	An expression in Python is evaluated for some results.
The execution of a statement changes the state of the variable.	The expression evaluation does not result in any state change.
A statement can be an expression.	An expression is not a statement.
Example : x=3*x*=3. **Output :** 33	**Example**: x=3+6*x*=3+6. **Output :** 99

Summary

- A combination of operands and operators is called an **expression**. The expression in Python produces some value or result after being interpreted by the Python interpreter. **Example: x=x+10**
- A statement is not evaluated for some result. A statement is used for creating variables or for displaying values.
- A statement can be an expression but an expression is not a statement.
- An expression in Python that contains only constant values are known as a **constant expression**.
- An expression in Python that contains a combination of operators, operands, and sometimes parenthesis is known as an **arithmetic expression**.
- An **integral expression** in Python is used for computations and type conversion (integer to float, a string to integer, etc.).
- A **floating expression** in Python is used for computations and type conversion (integer to float, a string to integer, etc.). A floating expression always produces a floating-point number as a resultant.
- A **relational expression** in Python can be considered as a combination of two or more arithmetic expressions joined using relational operators.
- A **logical expression** performs the logical computation and the overall expression results in either True or False (boolean result).
- The expression in which the operation or computation is performed at the bit level is known as a **bitwise expression** in Python.
- A **combination expression** can contain a single or multiple expressions which result in an integer or boolean value depending upon the expressions involved.

Indentation in Python

The spaces at the beginning of a code line are referred to as indentation. Whereas indentation in code is only for readability in other programming languages, it is critical in Python. Python employs indentation to denote a block of code. Indentation is an important concept in the Python programming language that is used to group statements that belong to the same block of code. In Python, a block of code is a group of statements that are executed together as a unit. Indentation in python is used to define the beginning and end of these blocks.

What is Indentation in Python?

In Python, indentation is the leading whitespace (spaces or/and tabs) before any statement. The importance of indentation in Python stems from the fact that it serves a purpose other than code readability. Python treats statements with the same indentation level (statements preceded by the same number of whitespaces) as a single code block. So, whereas in languages such as C, C++, and others, a block of code is represented by curly braces, a block in Python is a group of statements with the same Indentation level, i.e. the same number of leading whitespaces.

Rules of Indentation in Python

Here are the rules of indentation in python:

- Python's default indentation spaces are four spaces. The number of spaces, however, is entirely up to the user. However, a minimum of one space is required to indent a statement.
- Indentation is not permitted on the first line of Python code.
- Python requires indentation to define statement blocks.
- A block of code must have a consistent number of spaces.
- To indent in Python, whitespaces are preferred over tabs. Also, use either whitespace or tabs to indent; mixing tabs and whitespaces in indentation can result in incorrect indentation errors.

Example of Indentation in Python

Here are examples of indentation in python

Example 1
Below we have code implementation and explanation

- python

```
# your code goes here
# Python program showing

# indentation

site = 'prepbytes'

if site == 'prepbytes':

print('Logging on to prepbytes...')

else:

print('retype the URL.')

print('All set !')
```

Output

```
Logging on to prepbytes...

All set !
```

Explanation:
In the above python program, we have Print('Logging on to prepbytes…') and print('retype the URL.') are two distinct code blocks. In our example, if-statement, the two blocks of code are both indented four spaces. Because the final print ('All set!') is not indented, it is not part of the else block.

Example 2
Below we have code implementation and explanation

- python

```
# your code goes here

j = 1

while(j<= 5):

print(j)

j = j + 1
```

Output

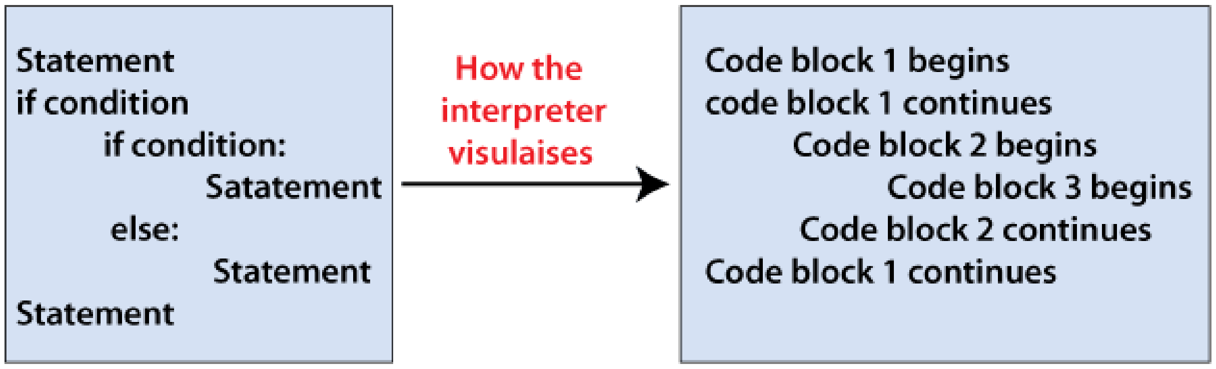
```
1
```

Explanation: In Python, you must indent each line of code by the same amount of whitespace to indicate a block of code. The while loop's two lines of code are both indented four spaces. It is required to indicate which code block a statement belongs to. For instance, j=1 and while(j=5): are not indented and thus do not fall within the Python while block. Indentation is used to structure Python code.

How do indentation errors Occur in Python?

If we are writing a certain code of lines based on some logic to get the desired result, but, meanwhile, we are following the above rules of indentation. The compiler will generate an indentation error.

Let us understand with the help of an example -



In the above figure, if we observe that it is mentioned how python recognises every block of code.

As we will notice in this figure, the indentation of every block is the calculation of whitespaces from the left corner to the statement began. Whenever the particular block of code starts, it will follow the **indentation.** At first, it will maintain the proper whitespaces and tabs, whenever the block starts and until it ends. If in the case inside that particular block includes a nested block, it will assign whitespaces to it till that nested block ends.

In this way, in the python programming language the indentation of a block matches.

An indentation error occurs whenever there is no ending of the particular block, or the number of whitespaces assigned for starting the code block is not the same. At the same time that blocks ends, ultimately, an indentation error occurs.

Description of the above figure -

- In the above figure, statement one will be executed. Then, the pointer will point to the ' if statement ' and check the condition; if the condition is true, it will enter the block and enter the nested if-else block.
- Again in the nested if statement, it will check the condition if it is true, then executes the next statement.
- In this way, the execution of the statements will be carried on, now; if we take a case where the block fails, it will not enter into the nested if-else block. It directly exit out form all the blocks and the final statement will execute, which matches with the block 1, initial block.

Advantages of Indentation in Python

Here are some advantages of indentation in python:

- **Improved Readability:** Indentation makes the code more readable by clearly indicating which statements belong to which block of code. This makes it easier for other developers to understand your code and makes it more maintainable.

- **Consistency:** By requiring consistent indentation levels, Python ensures that the code follows a consistent style, making it easier to read and understand.

- **Reduced Syntax:** Python's use of indentation reduces the amount of syntax required to define blocks of code. This makes Python code shorter and easier to read than code written in other languages.

- **Reduced Errors:** By requiring consistent indentation levels, Python reduces the likelihood of indentation-related errors that can occur in other languages. This improves code quality and reduces the time required for debugging.

- **Increased Efficiency:** Python's use of indentation simplifies the coding process and allows developers to write code faster and with fewer errors.

Disadvantages of Indentation in Python

Here are some disadvantages of indentation in python:

- **Limited Flexibility:** Python's use of indentation can limit the flexibility of code formatting, as all code blocks must follow a consistent indentation level. This can be problematic in certain situations where code formatting needs to be adjusted for different environments or requirements.

- **Difficulties with Copy and Paste:** If code is copied and pasted from one editor to another, the indentation levels may not be preserved. This can lead to errors in the code and make it more difficult to debug.

- **Increased Learning Curve:** The requirement for consistent indentation levels can increase the learning curve for new Python programmers. This can make it more difficult to get started with Python, especially for those who are used to other programming languages that use different syntax to define blocks of code.

- **Maintenance Issues:** In some cases, changes to the indentation levels can have unintended consequences that may be difficult to diagnose and fix. This can make maintenance more difficult and time-consuming.

How to fix indentation error in a python programming language?

There are points that one must remember while fixing the indentation error :

As we have seen earlier, indentation error occurs only due to the unmatched whitespaces and tabs; hence, we need to fix all the **whitespaces** and **tabs.**

- Firstly you need to find the error in a particular line of code.
- After that, check the number of whitespaces in that line and detect the block of code in which that line lies.
- Match the indentation of that block with the ending and starting point of that block by calculating the number of whitesces used in the starting and at the ending of that block.
- Still, if the error is not resolved, match the indentation of every block of code with another corresponding block of code by using the same way calculating the whitespaces and tabs used in it.
- Also, you take the help of the editor that you are using to check the number of whitespaces and tabs used, and you can easily detect from it by finding out the unnecessary whitespaces used.

Type Conversion in Python

Python defines type conversion functions to directly convert one data type to another which is useful in day-to-day and competitive programming. This article is aimed at providing information about certain conversion functions.

There are two types of Type Conversion in Python:

1. Implicit Type Conversion
2. Explicit Type Conversion

Let’s discuss them in detail.

Implicit Type Conversion

In Implicit type conversion of data types in Python, the Python interpreter automatically converts one data type to another without any user involvement. To get a more clear view of the topic see the below examples.

```
x = 10

print("x is of type:",type(x))

y = 10.6
print("y is of type:",type(y))

z = x + y

print(z)
print("z is of type:",type(z))
```

Output:

```
x is of type: <class 'int'>

y is of type: <class 'float'>

20.6

z is of type: <class 'float'>
```

As we can see the data type of ‘z’ got automatically changed to the “float” type while one variable x is of integer type while the other variable y is of float type. The reason for the float value not being converted into an integer instead is due to type promotion that allows performing operations by converting data into a wider-sized data type without any loss of information. This is a simple case of Implicit type conversion in python.

Explicit Type Conversion

In Explicit Type Conversion in Python, the data type is manually changed by the user as per their requirement. With explicit type conversion, there is a risk of data loss since we are forcing an expression to be changed in some specific data type. Various forms of explicit type conversion are explained below:

1. **int(a, base)**: This function converts **any data type to integer**. ‘Base’ specifies the **base in which string is** if the data type is a string.
   2. float(): This function is used to convert **any data type to a** floating-point **number.**

Example

```
# Python code to demonstrate Type conversion
# using int(), float()

# initializing string
s = "10010"

# printing string converting to int base 2
c = int(s,2)
print ("After converting to integer base 2 : ", end="")
print (c)

# printing string converting to float
e = float(s)
print ("After converting to float : ", end="")
print (e)
```

Output:

```
After converting to integer base 2 : 18

After converting to float : 10010.0
```

3. ord() : This function is used to convert a **character to integer.**
4. hex() : This function is to convert **integer to hexadecimal string**.
5. oct() : This function is to convert **integer to octal string**.

- Python3

```
# Python code to demonstrate Type conversion
# using ord(), hex(), oct()

# initializing integer
s = '4'

# printing character converting to integer
c = ord(s)
print ("After converting character to integer : ",end="")
print (c)

# printing integer converting to hexadecimal string
c = hex(56)
print ("After converting 56 to hexadecimal string : ",end="")
print (c)

# printing integer converting to octal string
c = oct(56)
print ("After converting 56 to octal string : ",end="")
print (c)
```

Output:

```
After converting character to integer : 52

After converting 56 to hexadecimal string : 0x38

After converting 56 to octal string : 0o70
```

6. tuple() : This function is used to **convert to a tuple**.
7. set() : This function returns the **type after converting to set**.
8. list() : This function is used to convert **any data type to a list type**.
Python3

```
# Python code to demonstrate Type conversion

# using tuple(), set(), list()



# initializing string

s = 'geeks'



# printing string converting to tuple

c = tuple(s)

print ("After converting string to tuple : ",end="")

print (c)



# printing string converting to set

c = set(s)

print ("After converting string to set : ",end="")

print (c)



# printing string converting to list

c = list(s)

print ("After converting string to list : ",end="")
print (c)
```

Output:

```
After converting string to tuple : ('g', 'e', 'e', 'k', 's')

After converting string to set : {'k', 'e', 's', 'g'}

After converting string to list : ['g', 'e', 'e', 'k', 's']
```

9. dict() : This function is used to **convert a tuple of order (key,value) into a dictionary**.
10. str() : Used to **convert integer into a string.**
11. complex(real,imag) : This function **converts real numbers to complex(real,imag) number.**
Python3

```
# Python code to demonstrate Type conversion

# using dict(), complex(), str()


# initializing integers

a = 1

b = 2


# initializing tuple

tup = (('a', 1) ,('f', 2), ('g', 3))


# printing integer converting to complex number

c = complex(1,2)

print ("After converting integer to complex number : ",end="")

print (c)


# printing integer converting to string

c = str(a)

print ("After converting integer to string : ",end="")
print (c)


# printing tuple converting to expression dictionary

c = dict(tup)

print ("After converting tuple to dictionary : ",end="")

print (c)
```

Output:

```
After converting integer to complex number : (1+2j)

After converting integer to string : 1

After converting tuple to dictionary : {'a': 1, 'f': 2, 'g': 3}
```

12. chr(number): This function **converts number to its corresponding ASCII character.**

Python3

```
# Convert ASCII value to characters

a = chr(76)

b = chr(77)


print(a)

print(b)
```

Output:

```
L

M
```