# UNIT - I

## Python History:-

- 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 |

**Tips to Keep in Mind While Learning Python:-**

1. Make it Clear Why We Want to Learn
2. Learn the Basic Syntax
3. Write Code by Own
4. Keep Practicing
5. Make Notes as Needed
6. Discuss Concepts with Other
7. Do small Projects
8. Teach Others
9. Explore Libraries and Frameworks
10. Contribute to Open Source

**Usage of Python:-**

- Desktop Applications
- Web Applications
- Data Science
- Artificial Intelligence
- Machine Learning
- Scientific Computing
- Robotics
- Internet of Things (IoT)
- Gaming
- Mobile Apps
- Data Analysis and Preprocessing

**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.

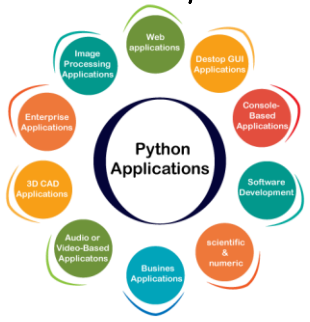
**1) Easy to Learn and Use**
Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language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

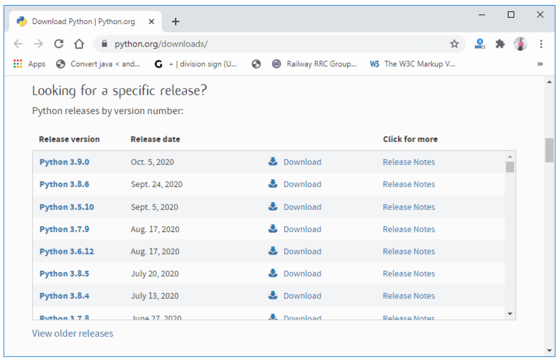
**Installation of Python (Environment Set-up):-**

In order to become Python developer, the first step is to learn how to install or update Python on a local machine or computer.

Visit the link *https://www.python.org/downloads/* to download the latest release of Python. In this process, we will install Python 3.8.6 on our Windows operating system.
When we click on the above link, it will bring us the following page.

**Step - 1: Select the Python's version to download.**

Click on the download button.



Looking for a specific release?

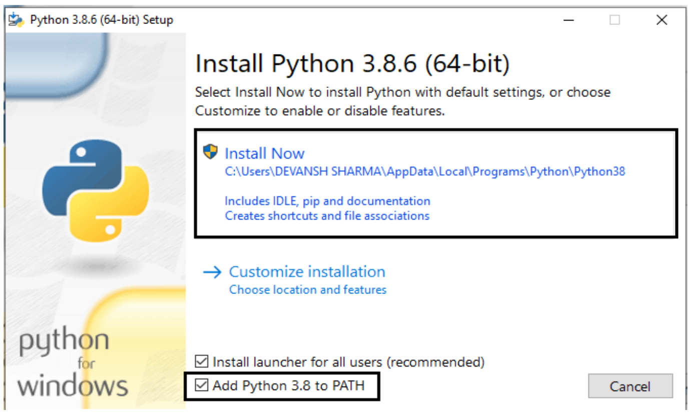
Python releases by version number:

| Release version | Release date | | Click for more |
|---|---|---|---|
| Python 3.9.0 | Oct. 5, 2020 | Download | Release Notes |
| Python 3.8.6 | Sept. 24, 2020 | Download | Release Notes |
| Python 3.5.10 | Sept. 5, 2020 | Download | Release Notes |
| Python 3.7.9 | Aug. 17, 2020 | Download | Release Notes |
| Python 3.6.12 | Aug. 17, 2020 | Download | Release Notes |
| Python 3.8.5 | July 20, 2020 | Download | Release Notes |
| Python 3.8.4 | July 13, 2020 | Download | Release Notes |
| Python 3.7.8 | June 27, 2020 | Download | Release Notes |

View older releases

**Step - 2: Click on the Install Now**

Double-click the executable file, which is downloaded; the following window will open. Select Customize installation and proceed. Click on the Add Path check box, it will set the Python path automatically.



Python 3.8.6 (64-bit) Setup

Install Python 3.8.6 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\DEVANSH SHARMA\AppData\Local\Programs\Python\Python38
Includes IDLE, pip and documentation
Creates shortcuts and file associations

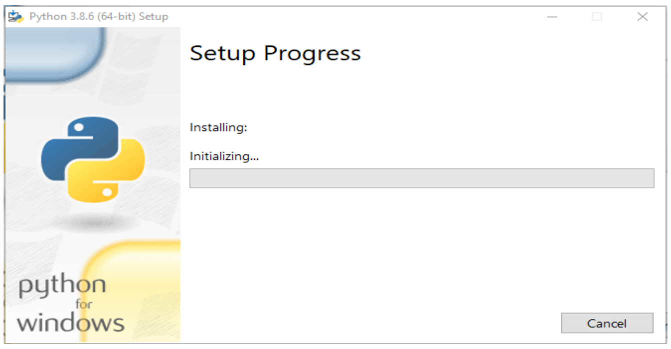
Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.8 to PATH

Cancel

We can also click on the customize installation to choose desired location and features. Other important thing is install launcher for the all user must be checked.

**Step - 3 Installation in Process**



Python 3.8.6 (64-bit) Setup

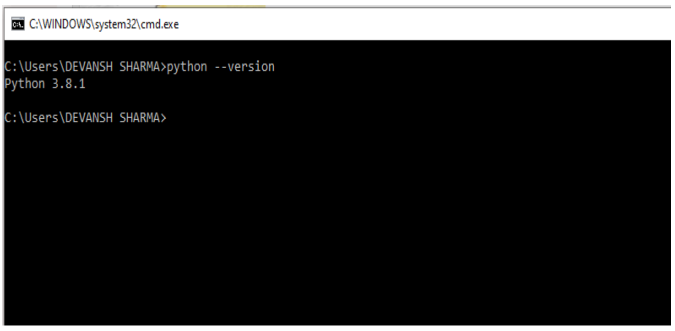
Setup Progress

Installing:

Initializing...

Cancel

Now, try to run python on the command prompt. Type the command python -version in case of python3.



```
C:\WINDOWS\system32\cmd.exe

C:\Users\DEVANSH SHARMA>python --version
Python 3.8.1

C:\Users\DEVANSH SHARMA>
```

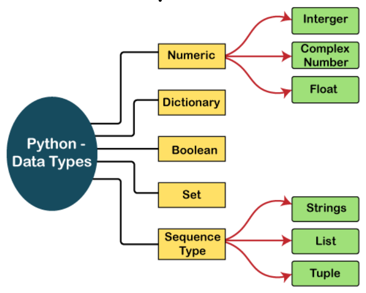
We are ready to work with the Python.

**Data Types:-**

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

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



**Python Variables:-**

- Variable is a name that is used to refer to memory location.
- Python variable is also known as an identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

**Identifier Naming:-**

Variables are the example of identifiers.

An Identifier is used to identify the literals used in the program.

The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore ( _ ).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive; for example, my name, and MyName is not the same.

Examples of valid identifiers: a123, _n, n_9, etc.

Examples of invalid identifiers: 1a, n%4, n 9, etc.

**Camel Case** - In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - nameOfStudent, valueOfVaraible, etc.

**Pascal Case** - It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.

**Snake Case** - In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

**Python Variable Types:-**

There are two types of variables in Python - Local variable and Global variable.

**Local Variables** - Local variables are the variables that declared inside the function and have scope within the function.

**Global Variables** - Global variables can be used throughout the program, and its scope is in the entire program. We can use global variables inside or outside the function.

**Delete a variable:-**

We can delete the variable using the **del** keyword. The syntax is given below.

**Syntax -** **del** <variable_name>

1. # Assigning a value to x
2. x = 6
3. **print**(x)
4. # deleting a variable.
5. **del** x
6. **print**(x)

**Tokens and their types:-**

- The tokens can be defined as a punctuator mark, reserved words, and each word in a statement.
- The token is the smallest unit inside the given program.

There are following tokens in Python:

- Keywords.
- Identifiers.
- Literals.
- Operators.

**Python Numbers:-**

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the type() function:

```
print(type(x))
print(type(y))
print(type(z))
```

**Type Conversion:-**

You can convert from one type to another with the int(), float(), and complex() methods:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
#convert from int to float:
a = float(x)
#convert from float to int:
b = int(y)
#convert from int to complex:
c = complex(x)
```

**Note:** You cannot convert complex numbers into another number type.

**Python String:-**

- Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes.

- The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.
- Each character is encoded in the ASCII or Unicode character.
- So we can say that Python strings are also called the collection of Unicode characters.

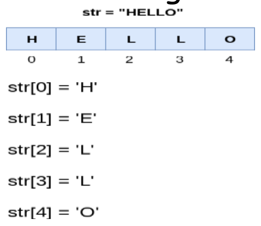
**Creating String in Python:-**

We can create a string by enclosing the characters in single-quotes or double- quotes. Python also provides triple-quotes to represent the string, but it is generally used for multiline string or **docstrings**.

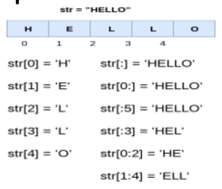
```
#Using single quotes
str1 = 'Hello Python'
print(str1)
#Using double quotes
str2 = "Hello Python"
print(str2)
#Using triple quotes
str3 = '''''Triple quotes are generally used for
represent the multiline or
docstring'''
print(str3)
```

**Strings indexing and splitting:-**

Like other languages, the indexing of the Python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.

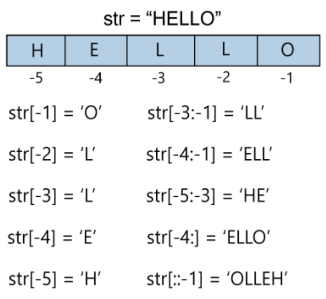


As shown in Python, the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in Python to access the substring from the given string. Consider the following example.



Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'HELLO' is given, then str[1:3] will always include str[1] = 'E', str[2] = 'L' and nothing else.

We can do the negative slicing in the string; it starts from the rightmost character, which is indicated as -1. The second rightmost index indicates -2, and so on. Consider the following image.



**Reassigning Strings:-**

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with new string since its content cannot be partially replaced. Strings are **immutable** in Python.

**Deleting the String:-**

As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

**String Operators:-**

| Operator | Description |
|---|---|
| + | It is known as concatenation operator used to join the strings given either side of the operator. |
| * | It is known as repetition operator. It concatenates the multiple copies of the same string. |
| [] | It is known as slice operator. It is used to access the sub-strings of a particular string. |
| [:] | It is known as range slice operator. It is used to access the characters from the specified range. |
| In | It is known as membership operator. It returns if a particular sub-string is present in the specified string. |
| not in | It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string. |
| r/R | It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string. |
| % | It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python. |

Consider the following example to understand the real use of Python operators.

1. str = "Hello"
2. str1 = " world"
3. **print**(str*3) # prints HelloHelloHello
4. **print**(str+str1)# prints Hello world
5. **print**(str[4]) # prints o
6. **print**(str[2:4]); # prints ll
7. **print**('w' **in** str) # prints false as w is not present in str
8. **print**('wo' **not in** str1) # prints false as wo is present in str1.
9. **print**(r'C://python37') # prints C://python37 as it is written
10. **print**("The string str : %s"%(str)) # prints The string str : Hello

**Python String Formatting:-**

**Escape Sequence**

- Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the **SyntaxError** as it contains both single and double-quotes.

- We can use the triple quotes to accomplish this problem but Python provides the escape sequence.
- The backslash(/) symbol denotes the escape sequence.
- The backslash can be followed by a special character and it interpreted differently.
- The single quotes inside the string must be escaped.
- We can apply the same as in the double quotes.

**Example:-**

```
# using triple quotes
print(''''They said, "What's there?"''')
# escaping single quotes
print('They said, "What\'s going on?"')
# escaping double quotes
print("They said, \"What's going on?\"")
```

The list of an escape sequence is given below:

| Escape Sequence | Description | Example |
|---|---|---|
| \newline | It ignores the new line. | print("Python1 \ <br>Python2 \ <br>Python3")<br>**Output:**<br>Python1 Python2 Python3 |
| \\ | Backslash | print("\\")<br>**Output:**<br>\ |
| \' | Single Quotes | print('\'')<br>**Output:**<br>' |
| \\'' | Double Quotes | print("\"")<br>**Output:**<br>" |
| \a | ASCII Bell | print("\a") |
| \b | ASCII Backspace(BS) | print("Hello \b World")<br>**Output:**<br>Hello World |
| \f | ASCII Formfeed | print("Hello \f World!")<br>Hello  World! |
| \n | ASCII Linefeed | print("Hello \n World!")<br>**Output:**<br>Hello<br> World! |
| \r | ASCII Carriege Return(CR) | print("Hello \r World!")<br>**Output:**<br>World! |
| \t | ASCII Horizontal Tab | print("Hello \t World!") |

| | | |
|---|---|---|
| | | **Output:**<br>Hello        World! |
| \v | ASCII Vertical Tab | print("Hello \v World!")<br>**Output:**<br>Hello<br> World! |
| \ooo | Character with octal value | print("\110\145\154\154\157")<br>**Output:**<br>Hello |
| \xHH | Character with hex value. | print("\x48\x65\x6c\x6c\x6f")<br>**Output:**<br>Hello |

Here is the simple example of escape sequence.

1. **print**("C:\\Users\\DEVANSH SHARMA\\Python32\\Lib")
2. **print**("This is the \n multiline quotes")
3. **print**("This is \x48\x45\x58 representation")

**Output:**

```
C:\Users\DEVANSH SHARMA\Python32\Lib
This is the
 multiline quotes
This is HEX representation
```

We can ignore the escape sequence from the given string by using the raw string. We can do this by writing **r** or **R** in front of the string. Consider the following example.

**print**(r"C:\\Users\\DEVANSH SHARMA\\Python32")

**Output:**

```
C:\\Users\\DEVANSH SHARMA\\Python32
```

**The format() method:-**

- The **format()** method is the most flexible and useful method in formatting strings.
- The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument.

1. # Using Curly braces
2. **print**("{} and {} both are the best friend".format("Devansh","Abhishek"))
3. #Positional Argument
4. **print**("{1} and {0} best players ".format("Virat","Rohit"))
5. #Keyword Argument
6. **print**("{a},{b},{c}".format(a = "James", b = "Peter", c = "Ricky"))

**Output:**

```
Devansh and Abhishek both are the best friend
Rohit and Virat best players
James,Peter,Ricky
```

**Python String Formatting Using % Operator:-**

- Python allows us to use the format specifiers used in C's printf statement.
- The format specifiers in Python are treated in the same way as they are treated in C. However, Python provides an additional operator %, which is used as an interface between the format specifiers and their values.
- In other words, we can say that it binds the format specifiers to the values.

Consider the following example.

```
1. Integer = 10;
2. Float = 1.290
3. String = "Devansh"
4. print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am strin
   g ... My value is %s"%(Integer,Float,String))
```

**Output:**

```
Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Devansh
```

**Python String functions:-**

Python provides various in-built functions that are used for string handling. Many String fun

| **Method** | **Description** |
|---|---|
| capitalize() | It capitalizes the first character of the String. This function is deprecated in python3 |
| casefold() | It returns a version of s suitable for case-less comparisons. |
| center(width ,fillchar) | It returns a space padded string with the original string centred with equal number of left and right spaces. |
| count(string,begin,end) | It counts the number of occurrences of a substring in a String between begin and end index. |
| decode(encoding = 'UTF8', errors = 'strict') | Decodes the string using codec registered for encoding. |
| encode() | Encode S using the codec registered for encoding. Default encoding is 'utf-8'. |
| endswith(suffix ,begin=0,end=len(string)) | It returns a Boolean value if the string terminates with given suffix between begin and end. |
| expandtabs(tabsize = 8) | It defines tabs in string to multiple spaces. The default space value is 8. |
| find(substring ,beginIndex, endIndex) | It returns the index value of the string where substring is found between begin index and end index. |
| format(value) | It returns a formatted version of S, using the passed value. |
| index(subsring, beginIndex, endIndex) | It throws an exception if string is not found. It works same as find() method. |
| isalnum() | It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false. |
| isalpha() | It returns true if all the characters are alphabets and there is at least one character, otherwise False. |
| isdecimal() | It returns true if all the characters of the string are decimals. |
| isdigit() | It returns true if all the characters are digits and there is at least one character, otherwise False. |
| isidentifier() | It returns true if the string is the valid identifier. |

| | |
|---|---|
| islower() | It returns true if the characters of a string are in lower case, otherwise false. |
| isnumeric() | It returns true if the string contains only numeric characters. |
| isprintable() | It returns true if all the characters of s are printable or s is empty, false otherwise. |
| isupper() | It returns false if characters of a string are in Upper case, otherwise False. |
| isspace() | It returns true if the characters of a string are white-space, otherwise false. |
| istitle() | It returns true if the string is titled properly and false otherwise. A title string is the one in which the first character is upper-case whereas the other characters are lower-case. |
| isupper() | It returns true if all the characters of the string(if exists) is true otherwise it returns false. |
| join(seq) | It merges the strings representation of the given sequence. |
| len(string) | It returns the length of a string. |
| ljust(width[,fillchar]) | It returns the space padded strings with the original string left justified to the given width. |
| lower() | It converts all the characters of a string to Lower case. |
| lstrip() | It removes all leading whitespaces of a string and can also be used to remove particular character from leading. |
| partition() | It searches for the separator sep in S, and returns the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings. |
| maketrans() | It returns a translation table to be used in translate function. |
| replace(old,new[,count]) | It replaces the old sequence of characters with the new sequence. The max characters are replaced if max is given. |
| rfind(str,beg=0,end=len(str)) | It is similar to find but it traverses the string in backward direction. |
| rindex(str,beg=0,end=len(str)) | It is same as index but it traverses the string in backward direction. |
| rjust(width,[,fillchar]) | Returns a space padded string having original string right justified to the number of characters specified. |
| rstrip() | It removes all trailing whitespace of a string and can also be used to remove particular character from trailing. |
| rsplit(sep=None, maxsplit = -1) | It is same as split() but it processes the string from the backward direction. It returns the list of words in the string. If Separator is not specified then the string splits according to the white-space. |
| split(str,num=string.count(str)) | Splits the string according to the delimiter str. The string splits according to the space if the delimiter is not provided. It returns the list of substring concatenated with the delimiter. |
| splitlines(num=string.count('\n')) | It returns the list of strings at each line with newline removed. |

|  |  |
|---|---|
| startswith(str,beg=0,end=len(str)) | It returns a Boolean value if the string starts with given str between begin and end. |
| strip([chars]) | It is used to perform lstrip() and rstrip() on the string. |
| swapcase() | It inverts case of all characters in a string. |
| title() | It is used to convert the string into the title-case i.e., The string **meEruT** will be converted to Meerut. |
| translate(table,deletechars = '') | It translates the string according to the translation table passed in the function . |
| upper() | It converts all the characters of a string to Upper Case. |
| zfill(width) | Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero). |
| rpartition() |  |

**Python print() Function:-**

The **print()** function displays the given object to the standard output device (screen) or to the text stream file.

Unlike the other programming languages, Python **print()** function is most unique and versatile function.

The syntax of **print()** function is given below.

**print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)**

Let's explain its parameters one by one.

**objects** - An object is nothing but a statement that to be printed. The * sign represents that there can be multiple statements.

**sep** - The **sep** parameter separates the print values. Default values is ' '.

**end** - The **end** is printed at last in the statement.

**file** - It must be an object with a write(string) method.

**flush** - The stream or file is forcibly flushed if it is true. By default, its value is false.

**Taking Input from the User:-**

- Python provides the **input()** function which is used to take input from the user.
- By default, the **input()** function takes the string input but what if we want to take other data types as an input.
- If we want to take input as an integer number, we need to typecast the **input()** function into an integer.
- We can take any type of values using **input()** function.

**Python Operators:-**

The operator is a symbol that performs a certain operation between two operands, according to one definition. In a particular programming language, operators serve as the foundation upon which logic is constructed in a programme. The different operators that Python offers are listed here.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators

- Bitwise Operators
- Membership Operators
- Identity Operators

**Arithmetic Operators:-**

| Operator | Description |
|---|---|
| **+ (Addition)** | It is used to add two operands. For example, if a = 10, b = 10 => a+b = 20 |
| **- (Subtraction)** | It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 15 |
| **/ (divide)** | It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0 |
| *** (Multiplication)** | It is used to multiply one operand with the other. For example, if a = 20, b = 4 => a * b = 80 |
| **% (reminder)** | It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0 |
| **** (Exponent)** | As it calculates the first operand's power to the second operand, it is an exponent operator. |
| **// (Floor division)** | It provides the quotient's floor value, which is obtained by dividing the two operands. |

**Comparison operator:-**

| Operator | Description |
|---|---|
| == | If the value of two operands is equal, then the condition becomes true. |
| != | If the value of two operands is not equal, then the condition becomes true. |
| <= | The condition is met if the first operand is smaller than or equal to the second operand. |
| >= | The condition is met if the first operand is greater than or equal to the second operand. |
| > | If the first operand is greater than the second operand, then the condition becomes true. |
| < | If the first operand is less than the second operand, then the condition becomes true. |

**Assignment Operators:-**

| Operator | Description |
|---|---|
| = | It assigns the value of the right expression to the left operand. |
| += | By multiplying the value of the right operand by the value of the left operand, the left operand receives a changed value. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30. |
| -= | It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10. |
| *= | It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200. |

| | |
|---|---|
| %= | It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0. |
| **= | a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a. |
| //= | A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a. |

**Bitwise Operators:-**

| Operator | Description |
|---|---|
| & (binary and) | A 1 is copied to the result if both bits in two operands at the same location are 1. If not, 0 is copied. |
| \| (binary or) | The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1. |
| ^ (binary xor) | If the two bits are different, the outcome bit will be 1, else it will be 0. |
| ~ (negation) | The operand's bits are calculated as their negations, so if one bit is 0, the next bit will be 1, and vice versa. |
| << (left shift) | The number of bits in the right operand is multiplied by the leftward shift of the value of the left operand. |
| >> (right shift) | The left operand is moved right by the number of bits present in the right operand. |

**Logical Operators:-**

| Operator | Description |
|---|---|
| and | The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b must both be true. |
| or | The condition will be true if one of the phrases is true. If a and b are the two expressions, then an or b must be true if and is true and b is false. |
| not | If an expression **a** is true, then not (a) will be false and vice versa. |

**Membership Operators:-**

| Operator | Description |
|---|---|
| In | If the first operand cannot be found in the second operand, it is evaluated to be true (list, tuple, or dictionary). |
| not in | If the first operand is not present in the second operand, the evaluation is true (list, tuple, or dictionary). |

**Identity Operators:-**

| Operator | Description |
|---|---|
| Is | If the references on both sides point to the same object, it is determined to be true. |
| is not | If the references on both sides do not point at the same object, it is determined to be true. |

**Operator Precedence:-**

The order in which the operators are examined is crucial to understand since it tells us which operator needs to be considered first. Below is a list of the Python operators' precedence tables.

| Operator | Description |
|---|---|
| ** | Overall other operators employed in the expression, the exponent operator is given precedence. |
| ~ + - | the minus, unary plus, and negation. |
| * / % // | the division of the floor, the modules, the division, and the multiplication. |
| + - | Binary plus, and minus |
| >> << | Left shift. and right shift |
| & | Binary and. |
| ^ \| | Binary xor, and or |
| <= < > >= | Comparison operators (less than, less than equal to, greater than, greater then equal to). |
| <> == != | Equality operators. |
| = %= /= //= -= += *= **= | Assignment operators |
| is is not | Identity operators |
| in not in | Membership operators |
| not or and | Logical operators |