CHAPTER 1

- INTRODUCTION TO PYTHON
- DATA TYPES AND OPERATORS
- CONTROL FLOW

Prepared by:
Afreen Banu, Ashwin R G

Python Tutorial

Learn Python

Python is a high-level, object-oriented, structured programming language with complex semantics. The high-level data structures coupled with dynamic typing and dynamic linking render it very appealing for Rapid Application Development and for use as a scripting or glue language to link established components. Here you will get a complete tutorial in Python, with all the topics required. Let’s start our Python tutorial.

Introduction of Python

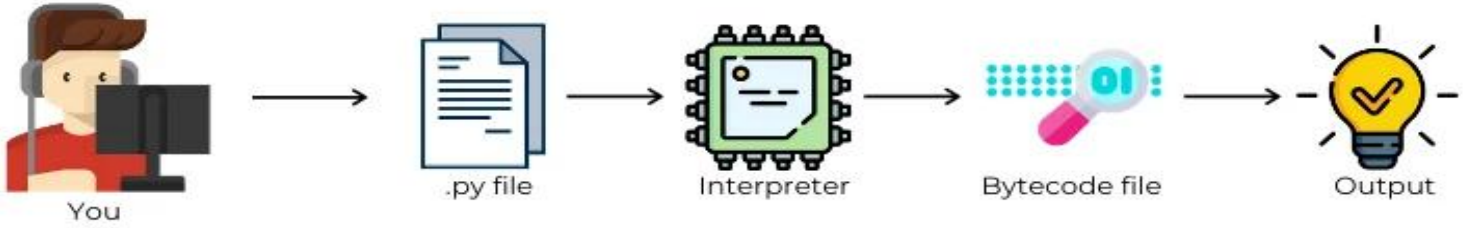
- What is Phython
- History of python
- What is bytcode in python
- Versions of python
- Implementation of python
    - What is Cpython?
    - What is Jython?
    - What is IronPython?
    - What id PyPy?

What is Python programming language?

Python

Python is an Open source, Free, High-level, Dynamic, and Interpreted programming language. Python was released in December 1989 by Guido van Rossum. Python programming language is one of the cleanest and easy to learn a language in the programming world. It’s easy syntax improved the readability of the code and makes it easy to understand.

What is Python?



The process to generate the output:

>> Write a high-level python code.
>> Save the code in .py file.
>> Interprete the code.
>> It will generate a bytecode file.
>>The output will get printed on the screen.

PrepInsta

Features of Python

ABC programming language is considered as the successor of the Python programming language. Guido Van Rossum who also contributed to the development of the ABC programming language referred the ABC language and improve the syntaxes of the ABC to generate a more clean easy and high-level programming language. Python has many features as it gets updates most frequently:

- Object-oriented
- Dynamically typed language
- Interpreted language
- Extensible
- Dynamic memory allocation
- Integrated

etc.

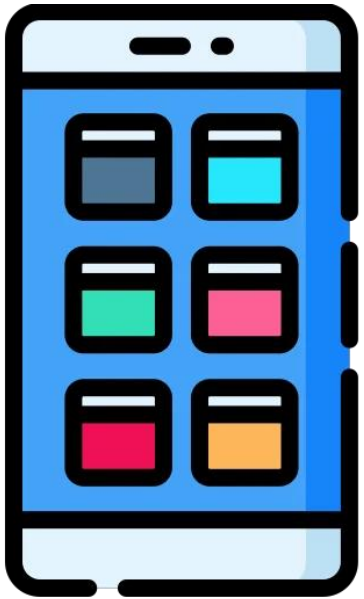
It is one of the most used language by programmers.

Applications of Python

Python programming language s used in many areas of development. Due to its easy understand ability and good user interface, it is used in many scientific fields. Many heavy operations Numeric and scientific calculations, Research and development, Scraping and many more. Some applications are:

- Machine Learning
- Artificial Intelligence
- Web Scraping
- Data Science and Analysis
- GUI

etc. Python has good library support that makes a programmer to write the program more easily and efficiently. It has library support for machine learning, Scraping, GUI, etc.



Companies using Python

Python is spreading in the air like fragrance of rose slowly but heavily. Today many techs. giants are adopting Python as their primary language and developing tools on python. Python's cross-platform support and its accessibility make programmers to code everything in python. Some big companies using python are:

- Instagram
- Spotify
- Amazon
- Google
- Netflix
- Dropbox

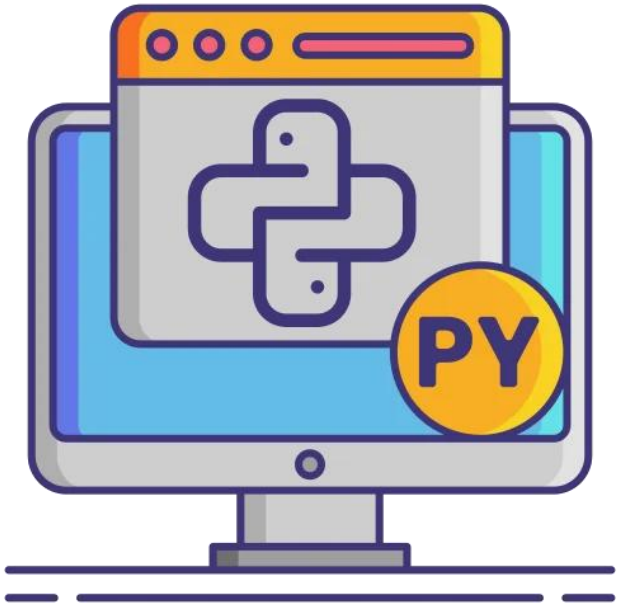
etc. Python is a good programming language to start, but once you start to write code in python it will remain forever.

History of Python

About the language

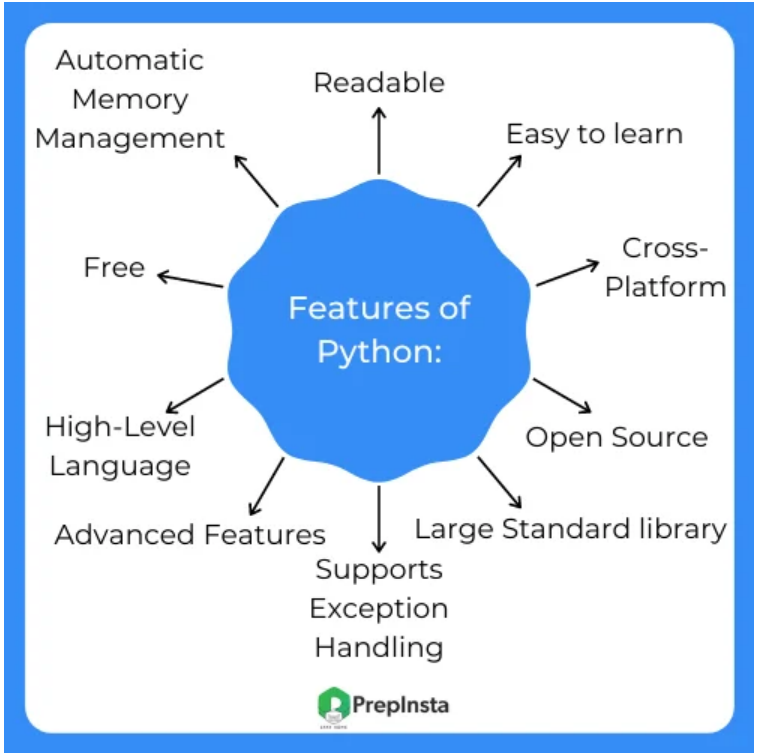
History of Python starts in the late 1980s at Centrum Wiskunde and Informatica, Python language was Designed by Guido Van Rossum and was first released in 1991.

Python language is the successor of ABC Language which had the feature of exception handling. Guido Van Rossum worked on the creation of ABC and so he took the syntax and features of the language, then solved all the issues with the language and created a good scripting language.



Features of Python:

- **Readable:** Syntax is similar to English and so the language is readable.
- **Easy to learn**: Python is expressive language and thus it is easy to learn.
- **Cross-Platform:** Python can run on different operating systems such as Windows, Mac, Linux, etc.
- **Open Source**: This is an open-source language.
- **Large Standard library:** Python has a very big library of pre-defined functions.
- **Free:** Python is freely available to use over your applications.
- Supports Exception Handling: Python has an advanced feature of exception handling.
- **Advanced Features**: Python supports many advanced features.
- **Automatic Memory Management**: Python supports this feature which automatically allocates and free the memory whenever used.



Why Python?

Python can run on different platforms like Windows, Mac, Linux, etc. It is easy to learn syntax as they are the same as the English language. Python codes are shorter than compared to other programming languages and hence are easy to write and debug. Python can be treated in an object-oriented, procedural, or functional way as it supports the concept of object-oriented programming and also functional programming. Some Special points in history of python are:

•It has a wast, active and supportive community.
•Sponsor of Python is Google.
•Huge library support.
•Efficient and easy to learn.
•Python is accessible.

Some applications of Python

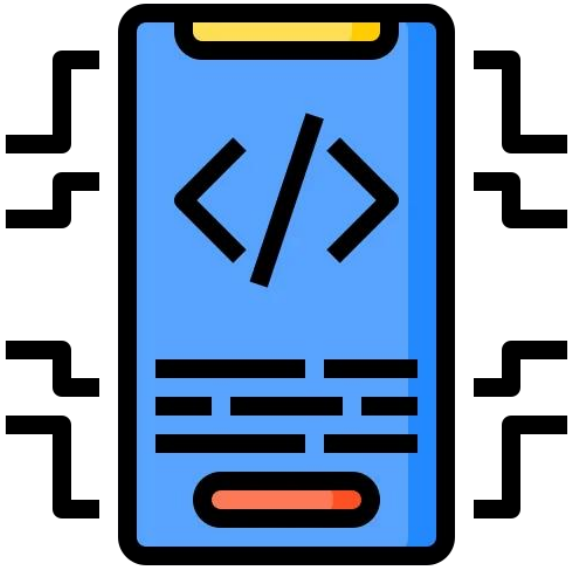
- Web Development
- Game Development
- Machine Learning
- Artificial Intelligence
- Data Science and Data Visualization
- Desktop GUI
- Web Scraping
- Business Applications
- Audio and Video Applications
- CAD Applications, etc.

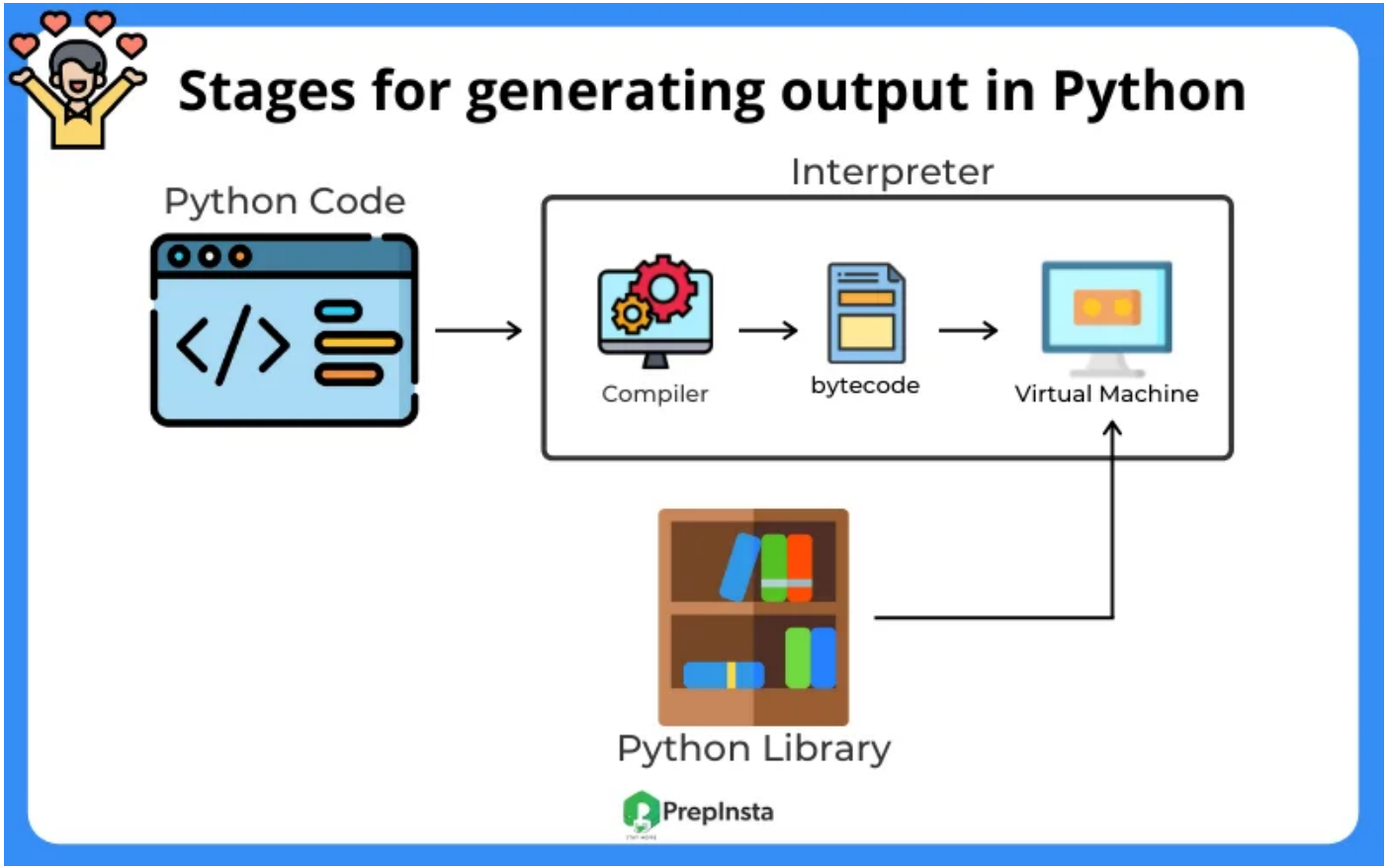
What is bytecode in Python

ByteCode

Machines are not able to understand the human language or high-level languages thus whenever we write code in any programing language, machine used a compiler or an interpreter to convert the human-readable code to machine code.

When the code is converted to machine code a new file is created which is easier for a machine to understand and execute. Such files contain code in bytes which is the machine-readable format. So let's discuss on the question more, What is bytecode in Python.




Stages for generating output in Python
Interpreter
Python Code
Compiler
bytecode
Virtual Machine
Python Library
PrepInsta

Byte codes are referred to as the portable codes or p-codes. When a python code is interpreted into the machine language then the python code gets converted into bytes.
These bytecodes are also called as the set of instructions for the virtual machine and the python interpreter is the implementation of the virtual machine. The set The intermediate format is called the bytecode.

Bytecodes are :
•Lower-level
•Platform Independent
•Efficient
•Intermediate
•Representation of source code

Bytecode is the low-level representation of the python code which is the platform-independent, but the code is not the binary code and so it cannot run directly on the targeted machine.
It is a set of instructions for the virtual machine which is also called as the Python Virtual Machine[PVM]. When the Python code is interpreted it is converted to the compiled bytecode file referred to as the PYC file. Pyc files have the set of instructions to follow in sequence to generate the output.
These pyc files are faster than the normal python code files

Bytecodes in pyc files

Python programing language is an interpreter language which converts the python code to the bytecode. These bytecodes are created by a compiler present inside the interpreter.
Interpreter first compiles the python code t0 the byte code which is also called as the intermediate code, then the code is used to run on the virtual machine.
In the virtual machine, the library modules of the python get added and then the code is ready to run on a machine.

Steps for interpretation of python source code:

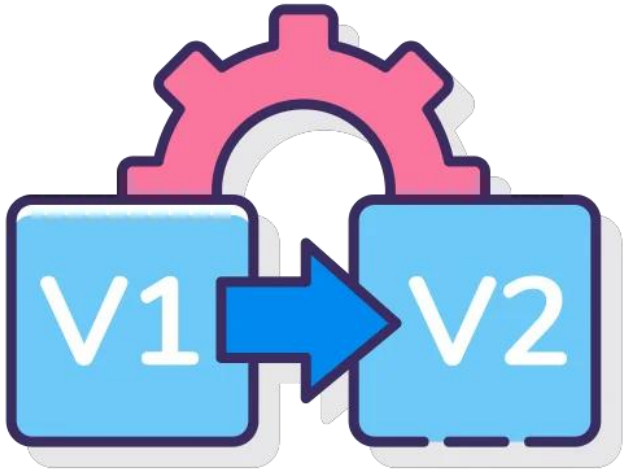
1.Source code: Python Code
2.Compiler: Enters inside the compiler to generate the bytecode
3.Bytecode: Intermediate code or low-level code
4.Virtual Machine: Here the code gets the support from the library modules.

Python is slower as compared to another programing language, but the process of converting the python code to the bytecode makes it faster to access each time after the code is interpreted once.
This bytecode is saved in the file named same as the source file but with a different extension named as “pyc”.

Versions of Python

Different versions of Python

Python was released in 1991 and till then we have seen many different version of python. initially, when Python was launched it was in its 0.9.0 version and now 3.8 is the most stable for the developers. Whereas Community already started working on 3.9 and 3.10. In total, we have seen 25 Versions of Python.



Different Versions of Python

Every version has some bugs in it. Fixing those bugs and updating the source code is called developing the new version.

Bugs like errors and also like not containing the updates which suits for latest applications. As we have talked in the above Paragraph about the 25 versions of python so lets now have a look over them and see how the language evolved.

Some important Versions of python

Versions of Python	Date of Release	Features
0.9.0	1991-02-20	Exception Handling, Class Inheritance, Data types like List, Str, and Dict
1.4	1996-10-25	Lambda, Map, Filter, Reduce, keyword argument, Built-in Support of Complex Numbers
2.0	2000-10-16	Garbage collection,List Comprehension, Data hiding by name mangling
2.7	2010-07-03	Pure object oriented approach, Generators, Warning Modes, and critical bugs were fixed
3.8	2019-10-14	Assignment Expressions,Positional-only Parameters, New and Improved Modules
3.9	2020-10-05	Coming Soon
3.10	2021-10-25	Coming Soon

Code Snippet for python version 2.7

```
print "Enter your name" name = raw_input() age = raw_input("age >>>") print "Your name is",name,"and","and your age is",age
```

Code Snippet for python version 3.8

```
name = input('Enter the name :')
age = int(input('Enter the age :'))
print('Name is ' + name + ' and age is ' + str(age))
```

What are different Implementations of Python

Implementations of Python programming

Python is a programing language introduced in 1991 by Guido van Rossum. Python is the successor of ABC programing language.

We have different implementations on the Python programming language. Implementation of the language is referred to as the program which provides support for the execution of a code written in python. Let's have a brief overview of implementations of python.

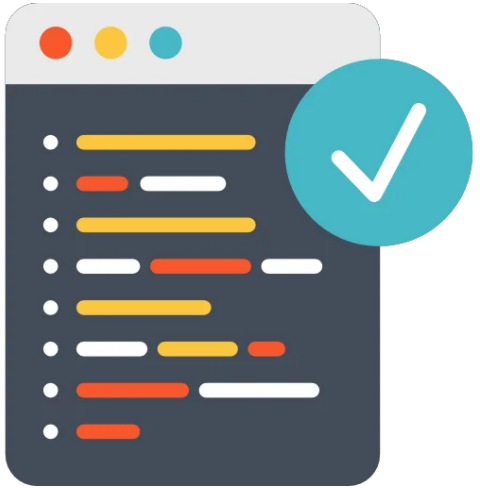
Implementations

We have many implementations of python programming in different languages. Some are:

- **CPython**
- **Jython**
- **PyPy**
- **IronPython**
- **Stackless Python**
- **MicroPython**

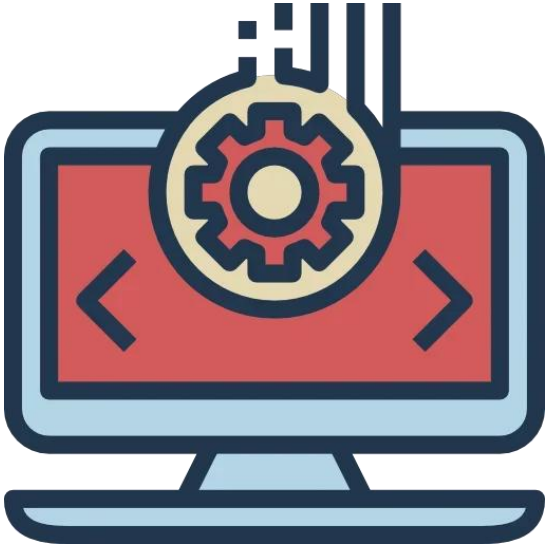
CPython

- CPython is the implementation of the Python programming language in C language. The prefix “C ” denotes C programming language.
- This is a compiled programming language that converts the python code to the bytecode by compiling it, and then give the code to the interpreter.
- CPython was introduced in 1994 by Python community sponsored by Python Software Foundation.
- It is faster than the original python programming language because it is a compiled language.
- It is used as the Scientific and numeric calculations and also for creating python standard modules.
- CPython follows the coding rules and syntax of python programming language versions 2.x.



Jython

- Jython is another implementation of python programming language. It was introduced on January 17, 2001, which written in java and python.
- Jython was designed to give support to the python programming language on the java platforms.
- Jython used most of the standard libraries of the python except for some modules written in C.
- Jython used classes of java instead of modules of the python programming language it can import and use any java class.
- Jython compiles python code to the java bytecode either on-demand or statically.
- The most recent release of Jython was Jython 2.7.2 on 21 March 2020 which is compatible with python 2.7.

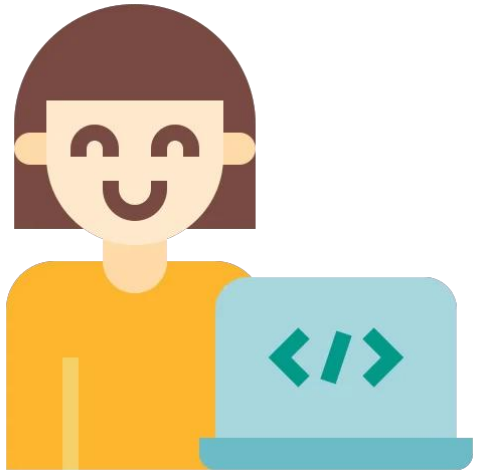


PyPy

- PyPy is an alternative implementation of Python programming language. It is Python interpreter and a compiler toolchain which is written in RPython. Although PyPy is faster than Python programming language as it used JIT[Just-in-time Compiler].

- Most of the python codes run well on this compiler except for some which are implemented in CPython. PyPy was first released in mid-2007 and got a stable release in 2020 as PyPy version 7.3.1. Meta-tracking is the technique used by PyPy which transforms an interpreter into a compiler also called as tracking just-in-time compiler.

- The current version of the PyPy is compatible with most operating systems such as Windows, Mac, Linux, etc.



IronPython

- IronPython is the implementation of python programming language which s targeting the .Net framework of Microsoft. The original author of the IronPython was Jim Hugunin who contributed in the initial version of the IronPython.

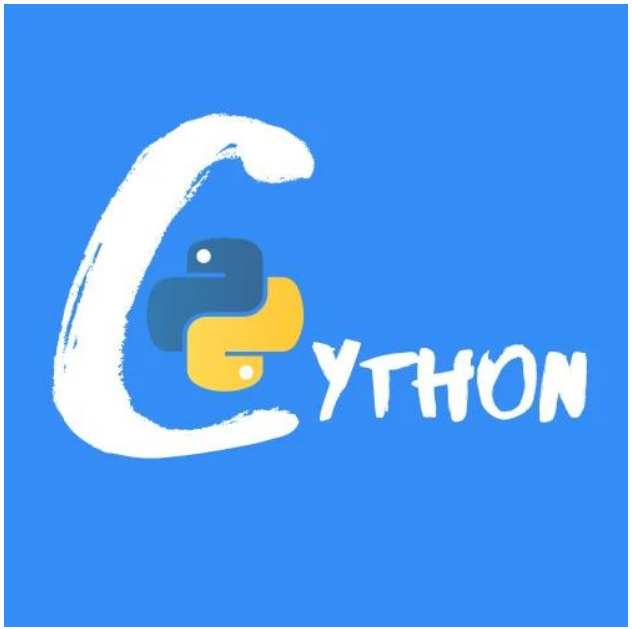
- The initial implementation of IronPython was released on Sept 5 2006. Then the project of IronPython was maintained by the small team at Microsoft until the launch of the version 2.7.

- The project is now maintained by the volunteers at GitHub. IronPython is written in C# also some of the code is generated by the generator written in python. Interface extensibility is the key advantage of IronPython.

What is CPython ?

What is Cython ?

Cython is the standard python software implementation in programming language C. Cython is both interpreted and compiled language, that is, before it is interpreted, it compiles the text into bytecode. In 1994 the first version of CPython was released by the Python developer community.



Applications of Cython

- Python programming will explicitly render calls to module C. Such C modules may be either generic C libraries or libraries specially designed for Python function.
- Cython produces the second form of the module: C libraries that refer to the internals of Python, and which can be combined with existing Python code. By default, Cython code looks very much like Python code.
- If you are feeding a Python program into the Cython compiler(Supporting both Python 2.x and Python 3.x), Cython must support it as-is, but neither of Cython 's native accelerations can come into effect.
- Yet if you decorate the Python code with the unique Cython syntax with form annotations, Cython would be able to substitute sluggish Python artifacts with fast C equivalents.

Uses of Cython

•Scientific and Numeric computations.

•High Traffic Websites.

•Designing Python Modules.

Python vs Cython vs Jython

Dimensions	Python	Cython	Jython
Introduction	Python is the traditional implementation of programming language, the one we probably use daily.	Cython is a superset of the Python programming language which additionally supports calling C/C++ functions.	Jython is mainly for integrating with the Java language.
Relation with Python Programming	Default Implementation	Alternative Implementation	Alternative Implementation
Compilation Process	compiles to .pyc	.pyx compiles to .c	compiles to .class
Extension	Extend with C	Extend with C	Extend with Java
Platform Dependency	Multi-platform	Multi-platform	100% Java
Garbage Collector	Python Garbage Collection	C Garbage Collection	Java Garbage Collection

Implementation of Cython

- To bring your Python into Cython, you must first build a file with the extension .pyx instead of the extension .py.
- You should start writing standard Python code within this file (note that there are certain restrictions in the Python code that Cython embraces, as explained in the Cython docs).
- Cython works by constructing a basic Python package.
- The action, though, varies from the Python standard in that the module code, initially written in Python, is converted into C.
- Although the resulting code is quick, several calls are made to the CPython interpreter and standard CPython libraries to do the actual work

Jython – Introduction

Jython :

Jython is a java implementation of python that combines expressive power with clarity.Jython is a powerful as well as simple because it uses Python's syntax and Java's environment.It allows using features of Python in Java environment or to import Java classes in Python codes and hence, it is very flexible.

Jython is freely available for both commercial and non-commercial use.

Advantages

- Embedded scripting – Java programmers can add the Jython libraries to their system to allow end users to write simple or complicated scripts that add functionality to the application.
- Interactive experimentation – Jython provides an interactive interpreter that can be used to interact with Java packages or with running Java applications. This allows programmers to experiment and debug any Java system using Jython.
- Rapid application development – Python programs are typically 2-10x shorter than the equivalent Java program. This translates directly to increased programmer productivity. The seamless interaction between Python and Java allows developers to freely mix the two languages both during development and in shipping products.

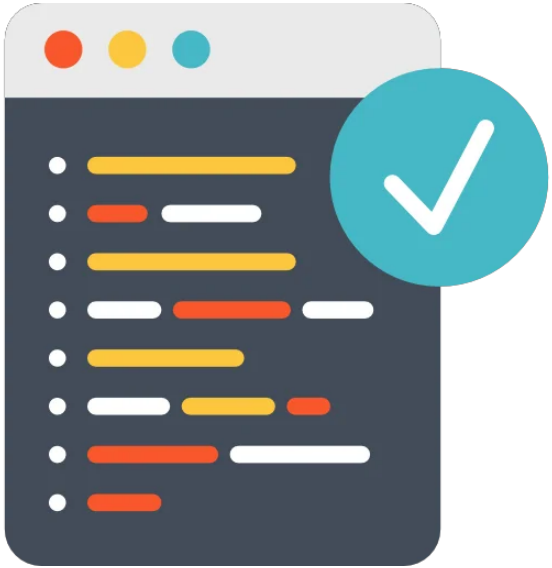
Example of Jpython :

```
#jpython Program
from java.lang import System # Java import

print('Running on Java version: ' + System.getProperty('java.version'))
print('Unix time from Java: ' + str(System.currentTimeMillis()))
```

IronPython-Introduction

IronPython :

IronPython is an excellent addition to the .NET Framework, providing Python developers with the power of the .NET framework. Existing .NET developers can also use IronPython as a fast and expressive scripting language for embedding, testing, or writing a new application from scratch.



Introduction :

- IronPython is written entirely in C#, although some of its code is automatically generated by a codegenrator written in Python.
- IronPython is implemented on top of the Dynamic Runtime Language (DLR), a library running on top of the Common Language Infrastructure that provides dynamic typing and dynamic method dispatch, among other things, for dynamic languages. The DLR is part of the .NET Framework 4.0 and is also a part of Mono since version 2.4 from 2009. The DLR can also be used as a library on older CLI implementations.

IronPyton vs CPython :

- There are some differences between the Python reference implementation CPython and IronPython. Some projects built on top of IronPython are known not to work under CPython. Conversely,
- CPython applications that depend on extensions to the language that are implemented in C are not compatible with IronPython , unless they are implemented in a .NET interop. For example, Numpy was wrapped by Microsoft in 2011, allowing code and libraries dependent on it to be run directly from .NET Framework.

PyPy – Introduction

What is PyPy :

PyPy is a very compliant Python interpreter, almost a drop-in replacement for CPython 2.7.10 and 3.3.5. It's fast due to its integrated tracing JIT compiler.

On average, PyPy is **4.2 times faster** than CPython. Sometimes Cpython may get TLE while execution but the same code may run fine with PyPy.In general if c++ takes 1 second to execute a file , if we execute same file with python it will take upto 5 seconds.

Advantages and distinct Features

- **Speed:** thanks to its Just-in-Time compiler, Python programs often run faster on PyPy.
- **Memory usage:** memory-hungry Python programs (several hundreds of MBs or more) might end up taking less space than they do in CPython.
- **Compatibility:** PyPy is highly compatible with existing python code. It supports cffi, cppyy, and can run popular python libraries like twisted, and django. It can also run NumPy, Scikit-learn and more via a c-extension compatibility layer.
- **Stackless:** It comes by default with support for stackless mode, providing micro-threads for massive concurrency.

Example #1:

```
#python Program
import timeit
start = timeit.default_timer()

#Your statements here
for i in range(10000):
    for j in range(1000):
        if(i==0 and j==0):
            print('In the loop')

print('Out of loop')
stop = timeit.default_timer()

print('Time: ', stop – start)
```

Output :

```
In the loop
Out of loop
Time: 0.912803540006628
```

Example #2:

#pypy Program

```
import timeit
start = timeit.default_timer()

#Your statements here
for i in range(10000):
    for j in range(1000):
        if(i==0 and j==0):
            print('In the loop')

print('Out of loop')
stop = timeit.default_timer()

print('Time: ', stop – start)
```

Output :

```
In the loop
Out of loop
Time: 0.021185765042901 04
```

A little more theory before Learning to Code

- Garbage collection in Python
- Dynamic Typing vs static typing Python
- Operator precedence and associativity of operators
- Why python is a strongly typed language?
- hex() functions in Python

Garbage collection in Python programming language

Garbage Collection

- Garbage collection is a tool used by the programming languages to delete the objects in memory that are not in the use.
- Garbage collection is invented by an American computer scientist John McCarthy in around 1959 to simplify manual memory management in lisp.
- Garbage collection can also be termed as the memory optimizer as it de-allocates the objects which are no longer in use in the code. Now lets us have a look at what is “Garbage collection in Python programming language”.

Strategies for memory management in Python

Python follows two approaches for memory management in the programming language:

- Reference Counting
- Garbage collection

These two features are used by the Python language for memory management. Now users don't have to use pre-allocate or de-allocate memory like dynamic memory allocation in languages like C, or C++.

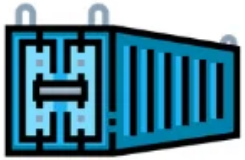
This feature was inherited by the ABC programming language which is considered as the successor of the Python programming language.

Garbage collection in Python programming language

Memory allocation

Initialize a local variable

variable = 1997 ⟶



Data stored in memory block

Garbage collection

When the scope of the variable is over

variable ⟶

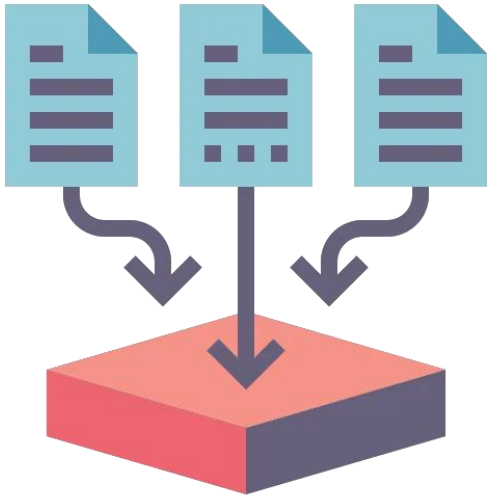
Memory de-allocation

PrepInsta

Reference Counting in Python programming language

In the python version, 2.x python uses Reference counting for memory management. Reference counting is done by counting the number of objects is referred by another object. As the reference of the object is removed the count of the object is de-allocated. As the reference count is decreased to 0 the object will be de-allocated.

The memory of the object or variable is freed only when the scope of the variable or object is over. Until the variable or an object is inside the scope it will not be cleared.



Garbage Collection in Python programming langua

Garbage collection in python programming as the automatic scheduled memory management tool. Garbage collection in python is scheduled upon a threshold of object allocation and object de-allocation. The garbage collection runs when the number of object allocation – the number of object de-allocation is greater than the threshold. The threshold of the garbage collection in python can be checked by some python commands.

```
import gc
print('Threshold is:', gc.get_threshold())
```

The above command will help you to know about the default threshold of the python. The garbage collection of the python will only run if the maximum threshold is crossed.

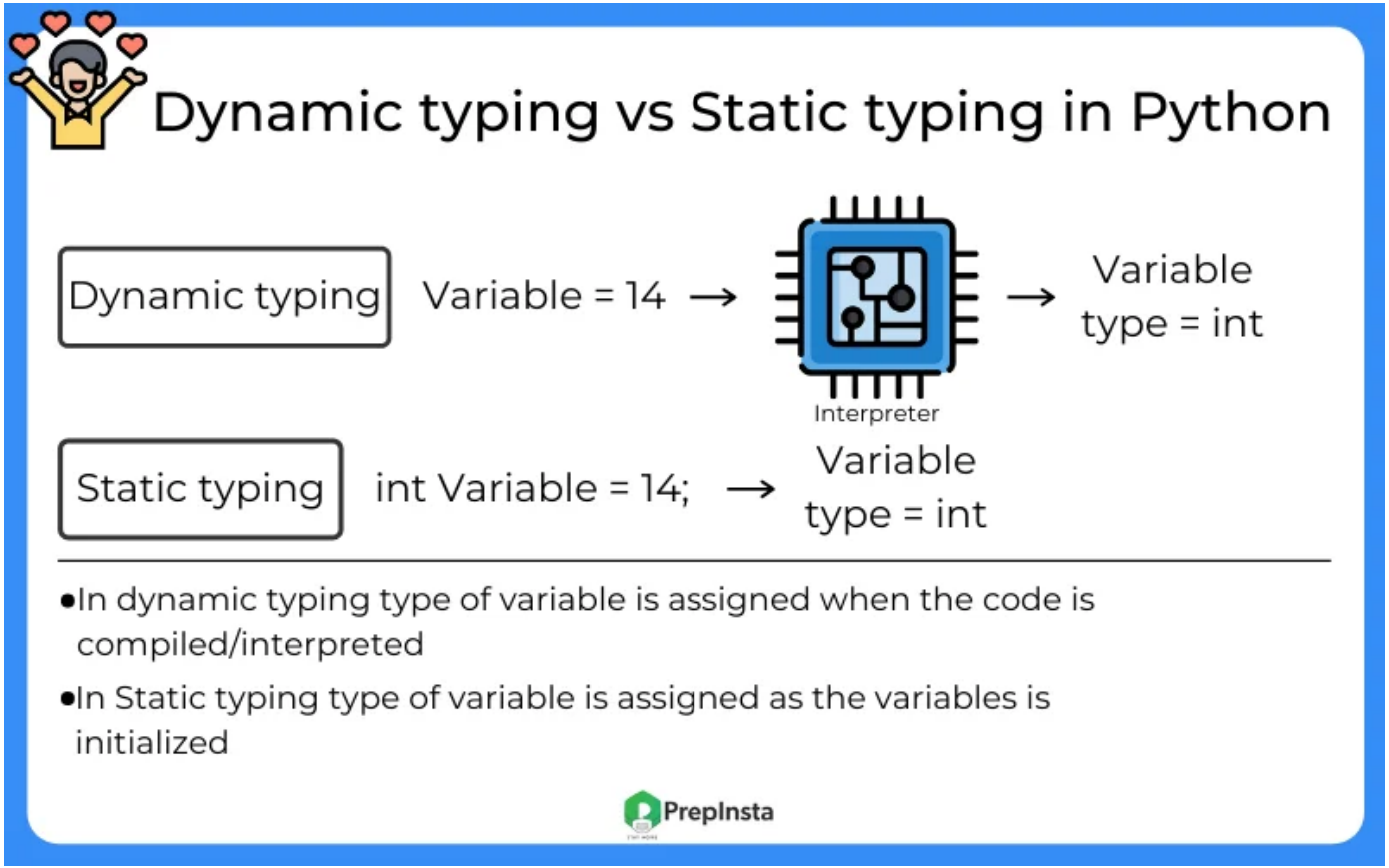
Dynamic Typing vs Static Typing in Python

Dynamic Typing vs Static Typing

- Dynamic typing and static typing are two different attributes of programing language on which they can be divided.
- Python is a dynamically typed language which means checking of the variable is done at the runtime. Whereas in the Statically typed language the checking of the variables or any other is done at the compile time.
- Dynamically typed language is easier to write as no need to initialize the type of the variable, but it also creates confusion while the code is revisited. Let’s see more on “Dynamic Typing vs Static Typing in Python”.

Difference between Dynamically and Statically typed language

Attribute	Dynamically Typed language	Statically Typed language
About	If the type of the variable is checked at the runtime of the code than the language is called a dynamically typed language.	If the type of the variable is checked at the compile-time of the code than the language is called a statically typed language.
Code Compilation/Interpretation	Code will get interpreted/compiled even if it contain error.	Code will not be compiled/interpreted until all the errors are corrected.
Variable deceleration	Variable without its type is initialized.	Variable with its type is initialized.
Languages	Python, PHP, Java-Scripy, etc.	Java, C, C++, etc.


Dynamic typing vs Static typing in Python
Dynamic typing
Variable = 14
Interpreter
Variable type = int
Static typing
int Variable = 14;
Variable type = int
•In dynamic typing type of variable is assigned when the code is compiled/interpreted
•In Static typing type of variable is assigned as the variables is initialized
PrepInsta

Dynamically Typed language

A language is considered as Dynamically typed language if the variable type of the language is checked at the runtime of the code compilation or code interpretation. In such type of programming languages, we don't need to initialize a variable with its type. We can declare a variable by writing the name at left and the value at the left of the variable name, Ex Var = 90. Some dynamically typed languages are:

- Python
- PHP
- JavaScript

As the memory allocation and variable checking are done at the runtime of the code, these type of languages are not considered as less optimised than statically typed language.

Statically typed language

In statically typed languages the type of the variable s checked at the compile time of the variable declaration. Statically programming languages check the type of the variable or abject while the code enters the compiler. Unlike dynamically typed languages we need to write the type of the variable during initializing it. Ex in java, int Var = 10. Some statically typed languages are:

- Java
- C
- C++

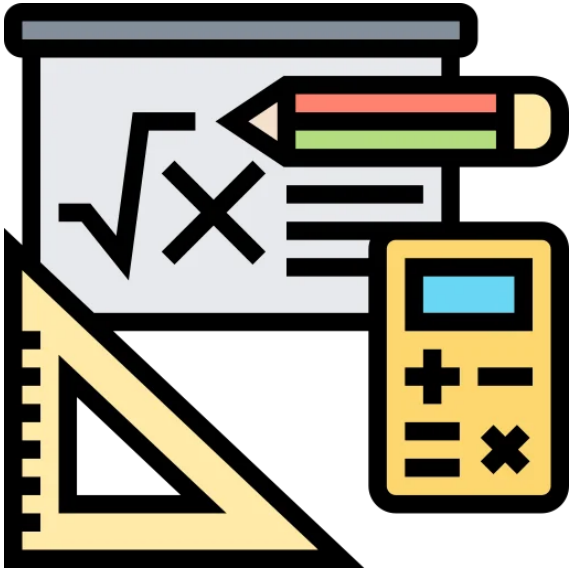
In Statically typed languages once if a variable is initialized to a data-type it cannot be assigned to the variable of a different type. Statically typed languages are faster than dynamically typed languages.

Operator Precedence and Associativity of Operators

Operator Precedence in Python

Operators are the key elements for performing any operation in any programming language. Operators tell the equation about the operation to be performed.

But what if we have a complex equation where more than one operator is paced between two operands and numbers of operands are more than 2. Ex equation = a+b-c*d/e. For these let's have a look at Operator Precedence and Associativity of Operators.



Different operators and their precedence

S.no	Operators	Name of the operator	Associativity orde
1	(,)	Brackets, Parenthesis	Left to Right
2	**	Exponent	Right to left
3	~	Bit-wise NOT	Left to Right
4	*, /, %, //	Multiplication, Division, Modulo, Floor division	Left to Right
5	+, –	Addition, Subtraction	Left to Right
6	>>, <<	Bit-wise right and left shift	Left to Right
7	&	Bit-wise AND	Left to Right
8	^	Bit-wise XOR	Left to Right
9	|	Bit-wise OR	Left to Right
10	=, !=, >, <, >=, <=	Comparison Operator	Left to Right
11	=, +=, -=, *=, /=, %=, **=, //=	Assignment Operator	Right to left
12	is, is not	Identity Operator	Left to Right
13	in, not in	Membership Operator	Left to Right
14	and, or, not	Logical Operator	Left to Right

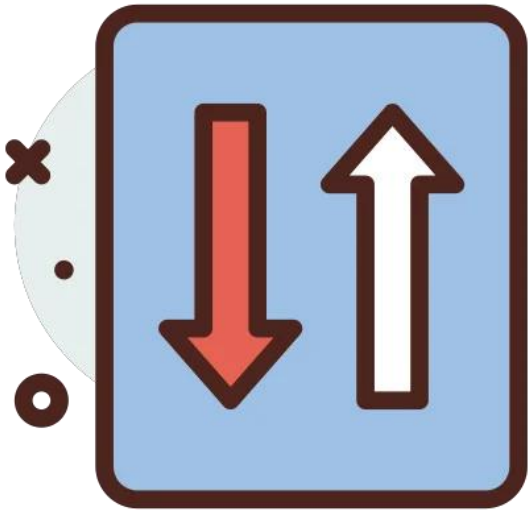
Above table shows the precedence and associativity of the operators, Precedence of the operator decrease as we move down in the table.

Precedence in operators

Precedence of the operator is to define which operator will be given higher priority to solve the equation. If in an equation more than one operator is present than the above table will be referred to perform operations. Different operators have different precedence and different usage. Advantages of following standard operator precedence table are:

- Same result.
- Follow universal grammar
- Simplicity
- Powerful approach

But what if we have 2 operators of the same precedence. These where associativity comes in the game.

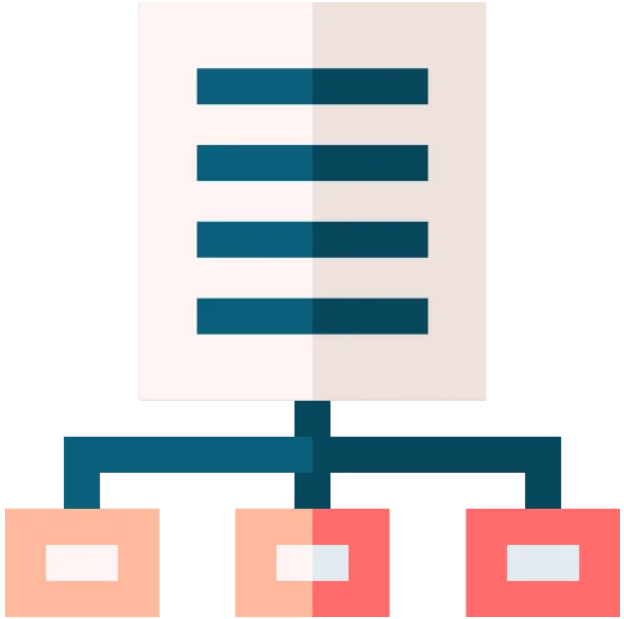


Associativity in Operators

Associativity of the operator is defined as in what order the operator is given Precision table should be used. Suppose in an equation we have both addition and subtraction operator, then which of the will be given higher precedence. Ex a+b-c, Some advantaged of following associativity table are:

- Simplify operator usage
- Powerful approach
- Universal fact

To understand this we refer the associativity table so we can solve more complex equations by following global approach.

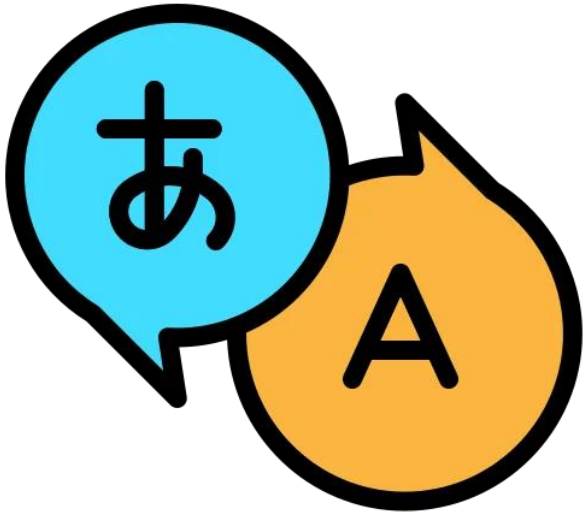


Why Python is a strongly typed language?

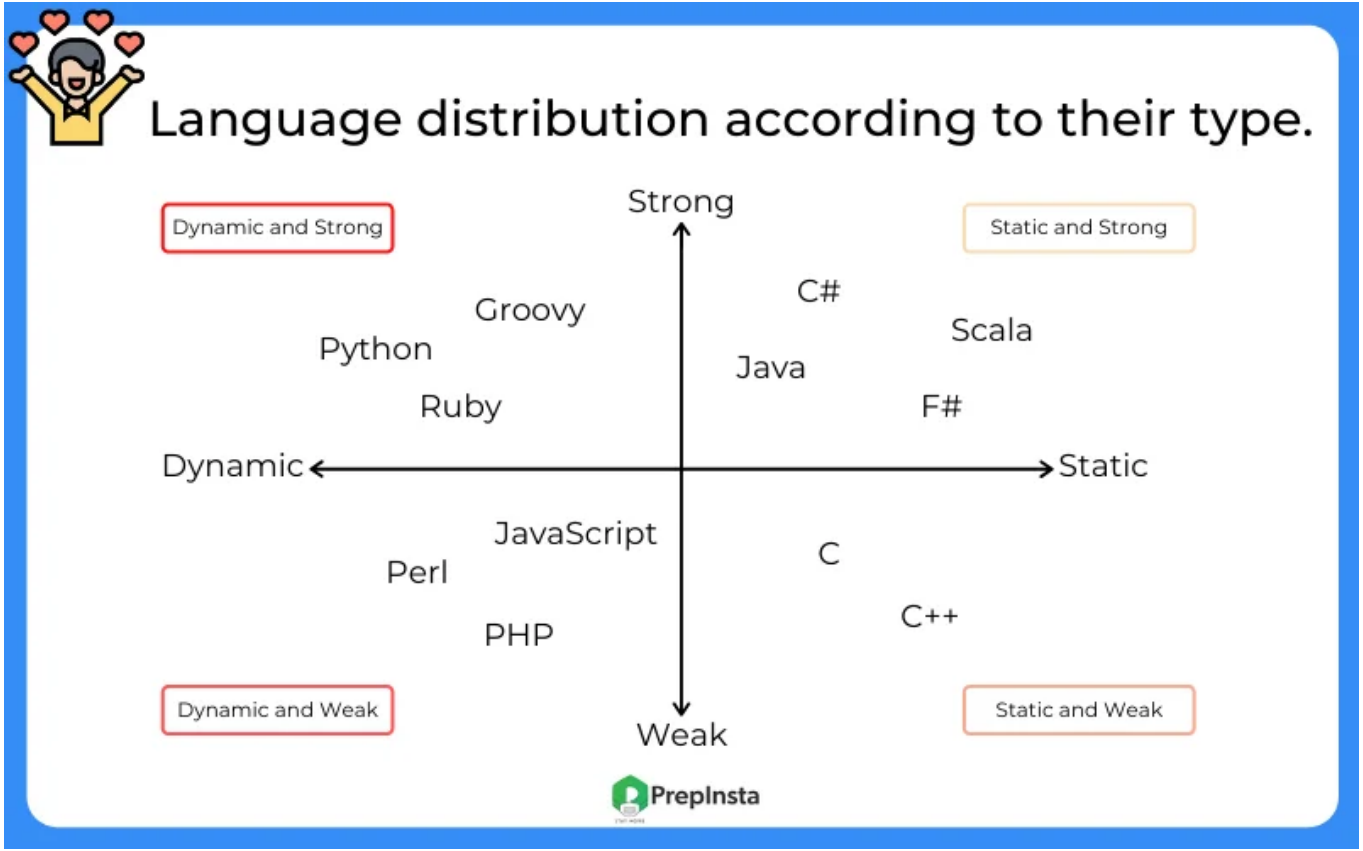
Strongly typed language

The programming language is referred to as the strongly typed language when the type of data like Integer, String, Float, etc are predefined. Also every variable or constant must be of the same data type.

A variable or a constant are not allowed without its type. As python is a dynamically typed language type of the variable is defines when the code is interpreted. Let's continue on the question "Why Python is a strongly typed language?".



Language distribution according to their type.



Why Python is a strongly typed language?

If a language is Strongly typed than it should follow some important rule that While the language is compiled or interpreted it must keep track of all the variables and constants that they are assigned to some data-type. In python programing language variables are not declared with the type of variable. But as the code interpreted the interprets keep track of all the type of variable in it.

Hence the variables initializes are assigned to the type of the category they belong it follows the most important rule of being a strongly typed language. And so we can say that the language is strongly typed.

#initialize some variables

```
var1 = 10
 var2 = 20.20
 var3 = 'PrepInsta'
#print type of these variables
print(type(var1))
print(type(var2))
print(type(var3))
```

Output:

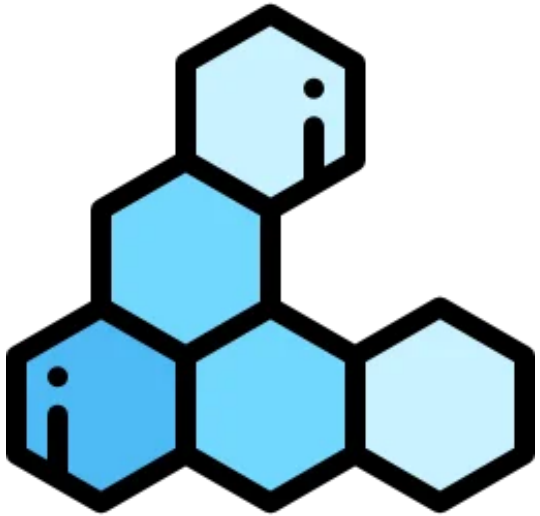
```
<class 'int'>
<class 'float'>
<class 'str'>
```

hex() function in Python programming language

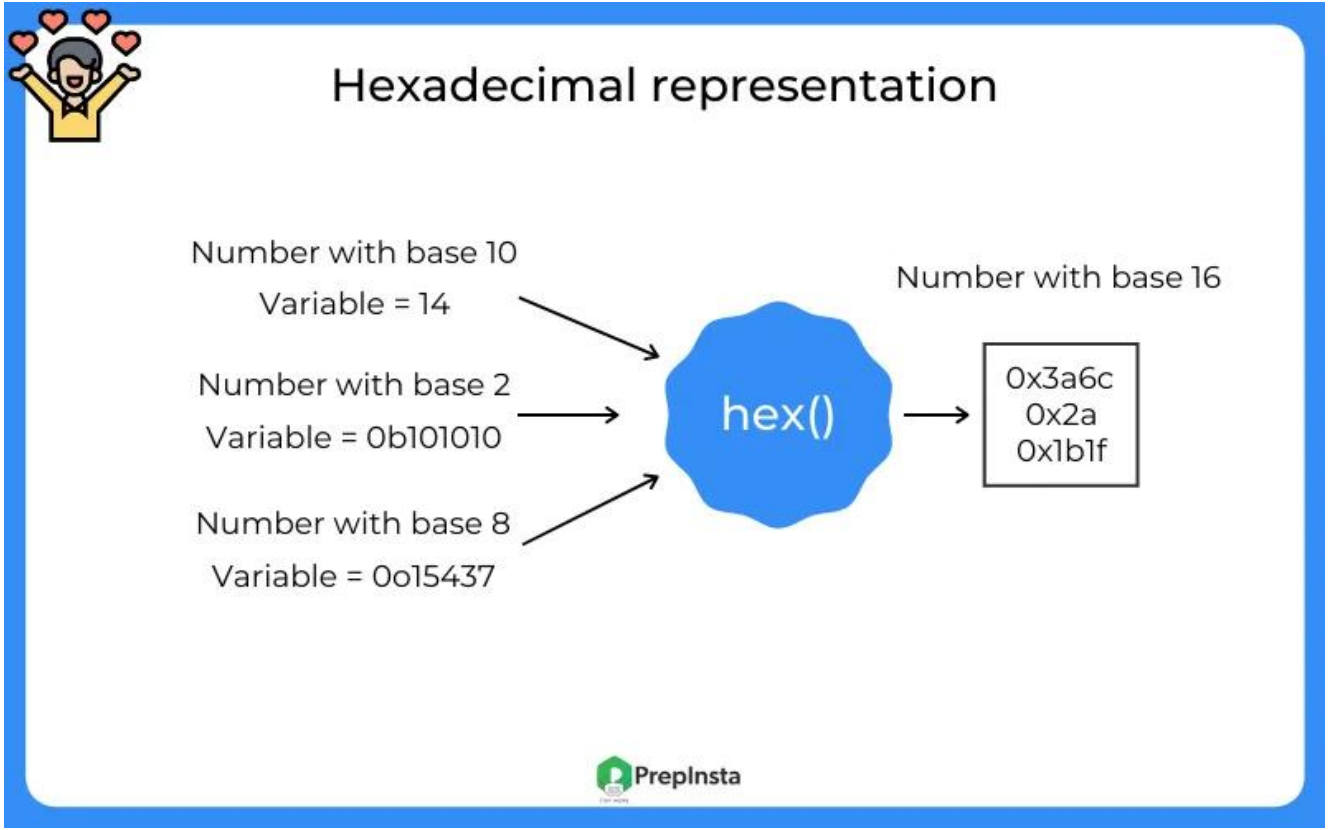
hex() function

hex() function in Python programming language is used to convert any type of representation of a number to hexadecimal. Hexadecimal numbers are represented with base 10.

Representation of these numbers is done in a unique way it used numbers from 0 to 9 to represent the first 10 numbers and then uses A to F alphabets to represent numbers from 10 to 15.



Hexadecimal representation



- Hexadecimal representation of a number is done using numbers from 0-9 and alphabets from A-F.
- Any representation of the number can be converted into hexadecimal by using hex() function.
- hex() function is an in-built function of python library.
- hex() function takes only one argument.
- As a representation of the number is passed in the function it returns the hexadecimal representation of the argument. Let’s have a look at python code to convert some number representation in hexadecimal format.

Python program to convert numbers to Hexadecimal format

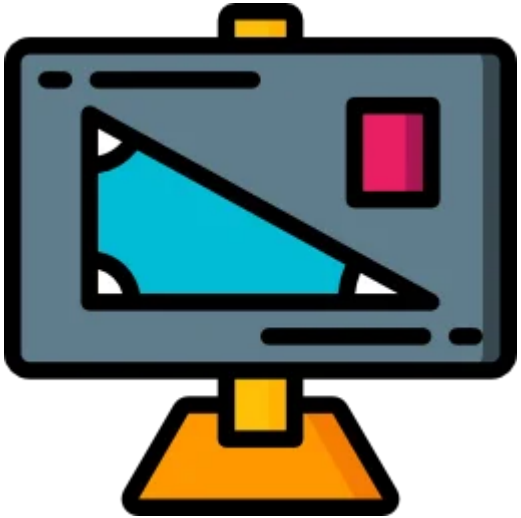
```
#integer number
 int_number = 14956
 #converting number to hexadecimal
 #using hex function
Hexa_number = hex(int_number) print(str(Hexa_number) + ' is hexadecimal representation of
number with base 10')

#binary number
bin_number = 0b101010
#converting binary number to hexadecimal
#using hex function
 Hexa_number = hex(bin_number)
print(str(Hexa_number) + ' is hexadecimal representation of number with base 2')

#octal number
oct_number = 0o15437
#converting octal number to hexadecimal
#using hex function
Hex_number = hex(oct_number)
print(str(Hexa_number) + ' is hexadecimal representation of number with base 8')
```

Output:

```
0x3a6c is hexadecimal representation of number with base 10
0x2a is hexadecimal representation of number with base 10
0x1b1f is hexadecimal representation of number with base 10
```

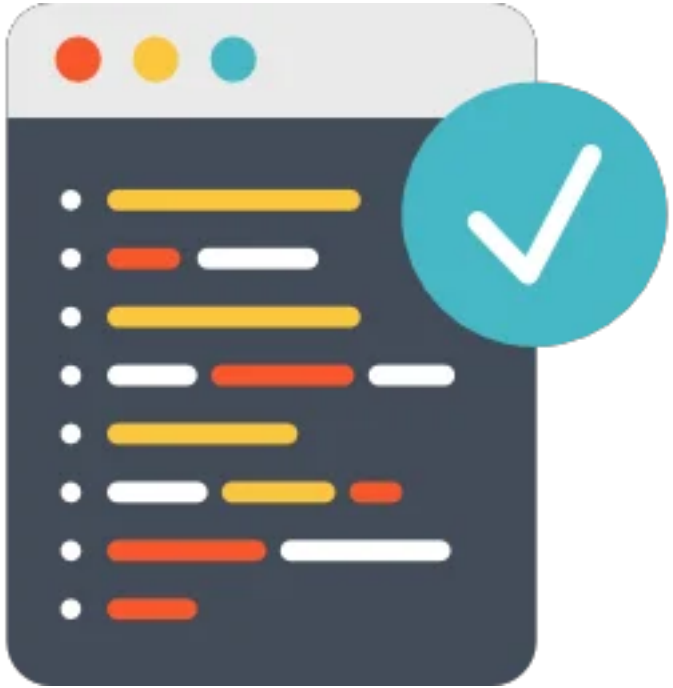
In the above python program where we converted numbers with base 10, 2 and 8 in the hexadecimal format or number with base 16 we used the in-built hex() function.

As the number enters the function it internally converts the number into the hexadecimal format and then using the print function we print it on the screen.



Data Types in Python

- Data Types in Python
- Ranges of data types in python
- Built-in functions in python
- Operators in python
- Numeric data type in python
- Str Datatype in python
- Iterator Data-types in python
- Sequence data types in python
- Blocks and statement in python
- Indentation in python
- Comments in python
- Bool() in python

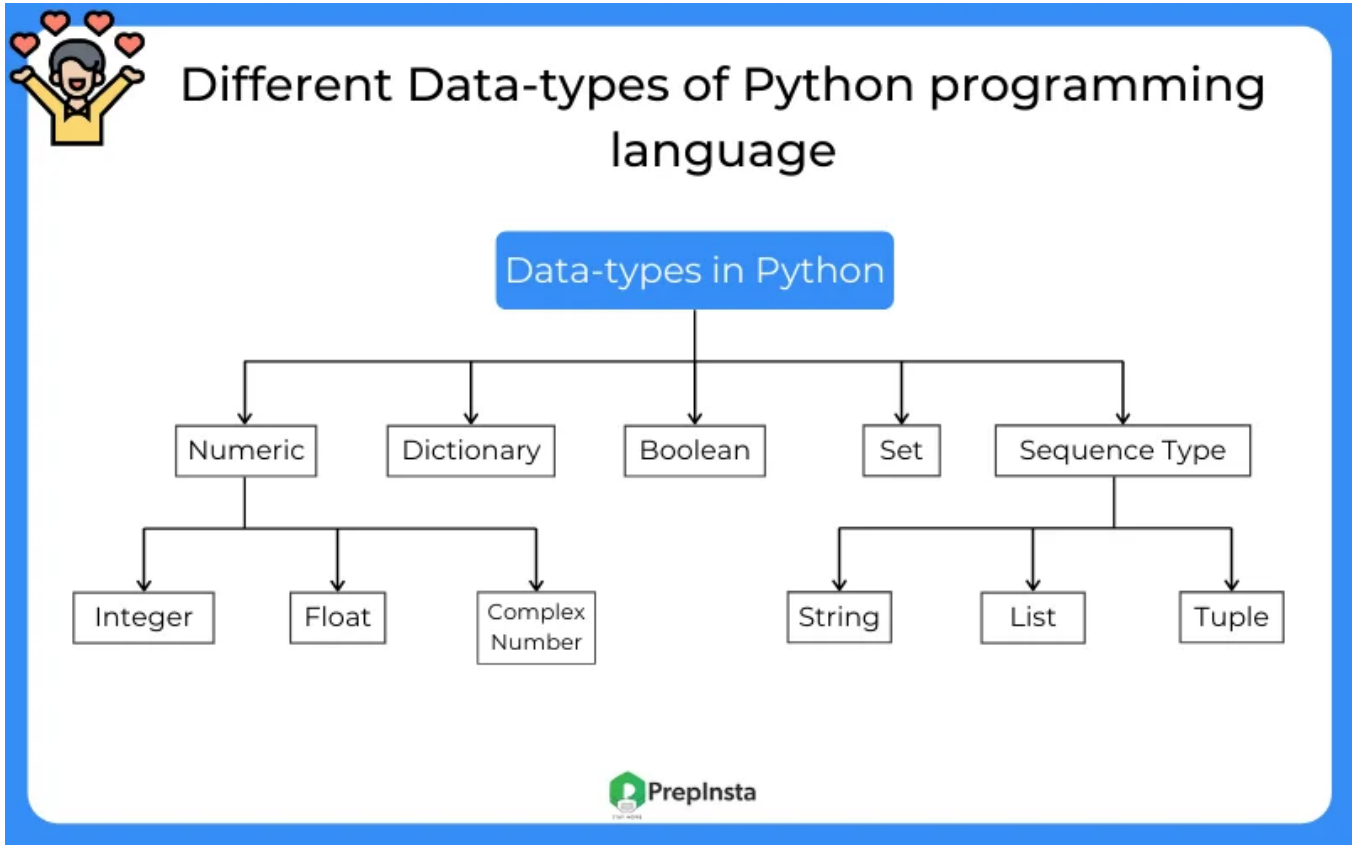


Data Types in Python programming language

Data Types in Python

While performing some operation or writing code in any programming language variables plays a major role. If you want to add two digits you can do it without any variable but what if the values are not known to you, in such case we use variables. Every variable can hold value of different type So let’s have a look at Data Types in Python programming language.

Different Data-types of Python programming language



1. Numeric:

Numeric Data-types are the data-type which represent the numeric values. It can be any number from the -infinity to +infinity. Numeric data types are one of them which can store values for an integer, Decimal values, Complex numbers, etc. Numeric data-types are subdivided into three types:

- Integer
- Float
- Complex Number

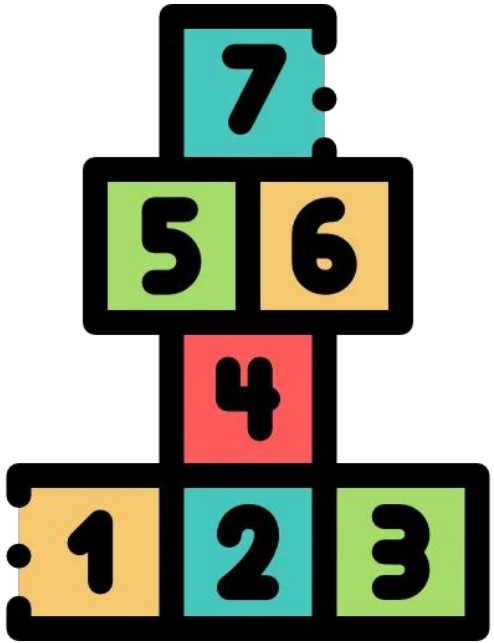
Integer

Integer data types are the one which can hold the integral value of the number. Integer data-types are the most common data types used in every programming language to deal with the operations performed on integer values. When we use loop statements to iterate through some list or any other data type, we use the integer values to get the indexes of the element.

```
#user input of integer value
var1 = int(input('Enter the integer value :'))
#initialize an integer variable
var2 = 2+56j
#print the integer variables
print(var1) print(var2)
```

Output:

```
Enter the integer value :
47
47
23
```



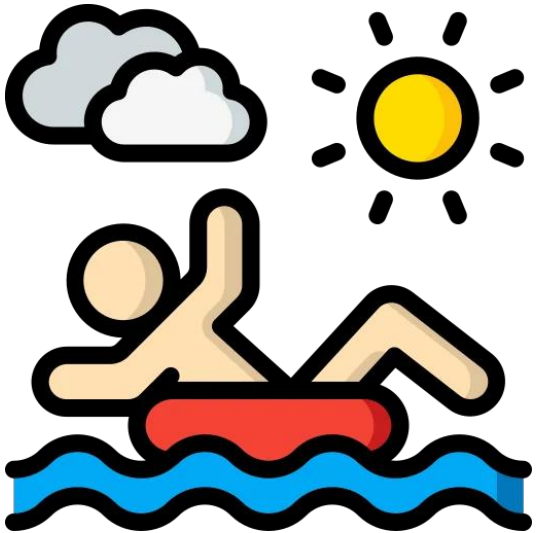
Float

Float Data-types is one which holds the decimal values written after an integral part of the number along with an integral part. Float data-type is used to store accurate values. They can be used defined inputs or obtained by performing some operations. Float data-types can store values up to some exact digits and also can be formatted according to users need.

```
#user input of float value
var1 = float(input('Enter the float value :'))
#initialize an float variable
var2 = 23.123123
#print the float variables
 print(var1) print(var2)
```

Output:

```
Enter the float value :
89.00012
89.00012
23.123123
```



Complex Number

Complex numbers are the data-type category that stores the complex numbers in a variable. Complex numbers are a combination of two different parts, one is a real part and another is the imaginary part. Complex numbers are written as Complex_number = 5+89j. Complex numbers are used to solve advance calculus problems.

```
#user input of complex value
var1 = complex(input('Enter the complex value :'))
#initialize an complex variable
var2 = 2+56j
#print the complex variables
print(var1)
print(var2)
```

Output:

```
Enter the complex value : 45+6j
(45+6j)
(2+56j)
```

Dictionary

Dictionaries are the unordered collection of data-types used to store the key-value pairs. Key in these dictionaries are the values that can be assigned to some data of any data-type, Whereas values are the information of the key. These key-value pairs can be accessed and used in many different ways.

Creating Dictionaries

Dictionaries in the Python Programming languages is the set of key holding some value to it and can be of any data-type. Dictionaries are created by placing the sequence of element inside the curly brackets {}, separated by commas. Values are the data stored in the key and can be duplicated, but keys are unique and duplicating keys are not allowed.

```
#creating dictionary
dictionary = dict({'Name': 'Divyanshu', 'Contact': 875597, 1:
'id_no'})
#printing dictionary
print(dictionary)
print(dictionary.keys())
print(dictionary.values())
```

Output:

```
{'Name': 'Divyanshu', 'Contact': 875597, 1: 'id_no'}
dict_keys(['Name', 1,  'Contact'])
dict_values(['Divyanshu', 875597, 'id_no'])
```

Boolean

Boolean data-type is used to store values with True and False. True value is stored the 1 and False stores the 0. Boolean data-types are used to check conditions. In conditional statements, boolean data-types are used to check if the value is correct or not. If the output is 1 or true it means the condition is satisfied and if the value is 0 or false it means the condition is not satisfied.

Boolean data-types are used to check the condition whether it is TRUE or FALSE. To check any condition we used if-else statements and loops. If the condition is True then the code inside the condition will get executed otherwise the code will stop and come out of the loop.


```
#Boolean in is-else conditions
if True:
   print('I am always True')
else:
   print('I am False')
#boolean in while loop
i = 1
while i == True:
   print('I am True')
   i = 0
```

Output:

I am always True
I am True

Set

Sets are Data-type like the list. In this type of data type, we can store different values which are non-repeating. Sets remove the duplicates of the elements and rearrange them in an unpredictable arrangement. We cannot define the position of the element in the set but sets help find if the element is available in it or not.

Sets are used when we need to search if the item is available or not. It is the optimized version of the list in some ways. List and sets both can store values but in sets duplicate values are removed and operations are performed. Sets are also iterable and mutable and contain a huge set of elements.

```
#user defined set
Set = set([1,2,3,4,3,1,6])
print(Set)
#check for element
if 1 in Set:
    print('True')
```

Output:

{1, 2, 3, 4, 6}

True

Sequence

The sequence in a python programming language is a collection of data of same data-type or different data-type. In sequences, we can store items or elements and can efficiently perform operations. The sequence elements are used in performing various operations. The sequence is subdivided in three parts:

- List
- String
- Tuple

List

The list is a subpart of the datatype sequence. Lists are used to store values of same and different data types like integer, string, float, etc. The list is one of the most efficient and mutable data-type in the python programming language. The list can perform several operations like append the new elements, remove the item, reverse the elements, etc. All these functions are available in the python library.

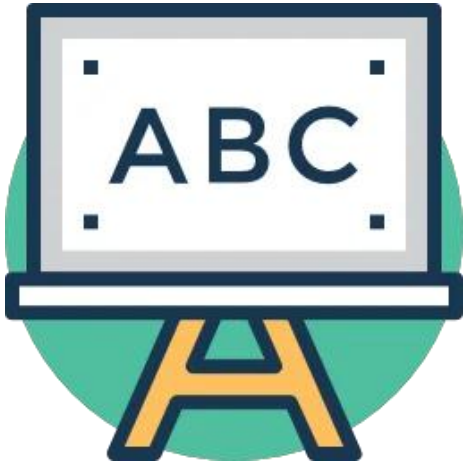
```
#initializing a list
List = [1,2,3,4,2,1]
#print the list elements
print(List)
#Printing type of sequence
print(type(List))
```

Output:

```
[1, 2, 3, 4, 2, 1]
<class 'list'>
```

Strings

Strings in python programming is a subpart of sequence data type which contains character values. The string is a very-strong data-type used in python to perform operations. It is immutable data-type but has the property of iterations. We can perform many different operations on strings. In python, there is no datatype as a character, and a character is treated as the string of length 1.



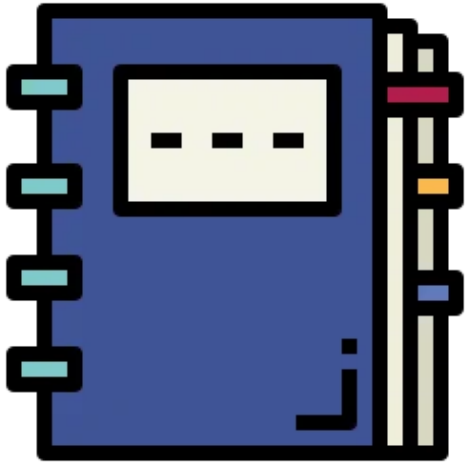

```
#initializing an string
STRING = "Have a Good Day"
print(STRING)
#user defined strings
STR = str(input('Enter the string :'))
print(STR)
```

Output:

```
Have a Good Day
Enter the string :PrepInsta
PrepInsta
```

Tuple

Tuples belong to the data-type class classed as a sequence. Like List and set, a tuple can also store elements of same or different data-type. The tuple is an immutable collection of elements. we can perform operations on tuple but we cannot remove or append elements in the tuple. Tuples are hashable whereas lists are not. Tuples elements are enclosed in brackets ().



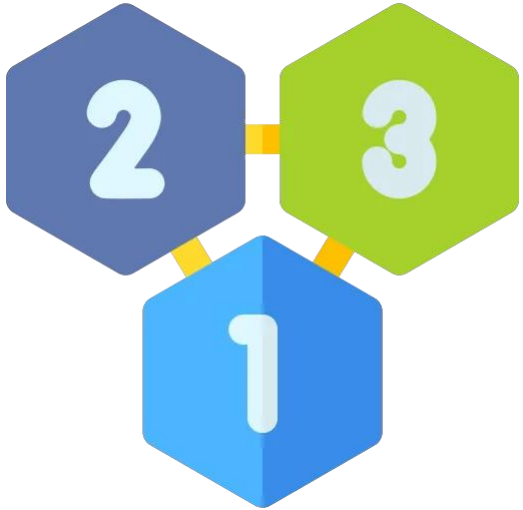
```
#initializing a tuple
Tuple = tuple([1,2,3,1,2])
print(Tuple)
#Printing type of sequence
print(type(Tuple))
```

Output:

```
(1, 2, 3, 1, 2)
<class 'tuple'>
```

Ranges of Data Types in Python

Range of Data Types :
Data types are the classification of data items. It represents that kind of value which tells what operations can be performed on a particular data type.Everything is an object in Python programming, data types are actually classes and variables are instance of these classes.



Example :

- Integer – 4, 5 ,6
- Float – 3.22 , 0. 0111

In python , value of an integer/float is not fixed and it can be as large as possible until our memory has some space to accomodate it. So we never need any special keyword for large integer or float like c/c++ have.
Hence python 3 has only integer and float , but in python 2.7 there was two types of integer ‘int’ and ‘Long int’.

Code #1:

```
#python Program

#Range on data type

x=99999999999999999999999999999999999999999999999
99999999999999999999
print('X is = ' , x)

y=0.00000000000000000000000000000000000000000000
0000000000000000000000001

print('Y is = ' , y)
```

Output:

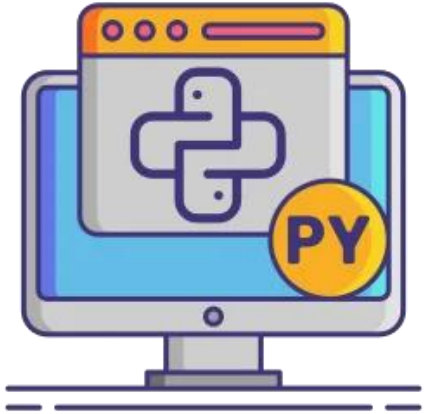
X is = 99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999

Y is = 1e-69

Built-in Functions in Python

What are Built-in functions?

Features which are given as part of a high-level language that can be implemented through a basic connection with statement definition are known as Built-in Functions in Python. Python offers a broad range of built-in functions which makes it more versatile, easy to use, and fast to develop. Due to this broad range of built-in functions, you can do more with less code.

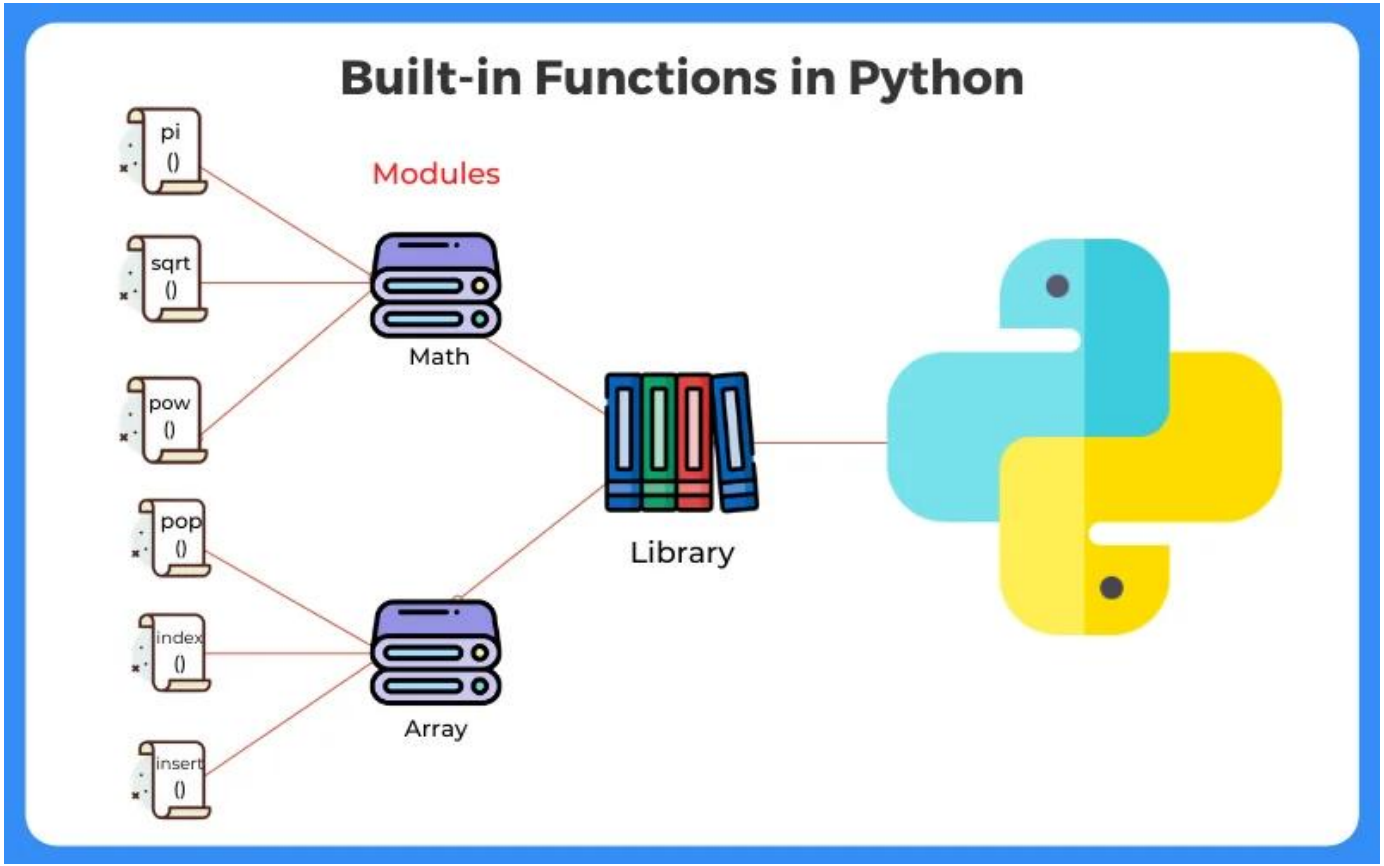


Some of popular Built-in Functions in Python:

Functions	Description	Example
int()	Return an integer object constructed from a number or string *x*, or return `0` if no arguments are given	>>> int('23') 23
len()	Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).	>>>len([1,2,3,4,5]) 5
reversed()	returns a reverse iterator	>>> for i in reversed([1,2,3,4,5]): print(i,end=", ") 5, 4, 3, 2, 1,
float()	Return a floating point number constructed from a number or string x. If no argument is given, **0.0** is returned.	>>>float('+12.34') 12.34 >>> float(123) 123.0

round()	Return *number* rounded to *ndigits* precision after the decimal point.
	If ndigits is omitted or is **None**, it returns the nearest integer to its input.
		>>> round(2.122345,3)
		2.122
		>>> round(12.32)
		12

input()	Function reads a line from the input, converts it to a string, and returns it.	>>> input() asdf 'asdf'
sorted()	Return a new sorted list from the items in *iterable*.	>>> sorted([1,4,6,3,2,8,4]) [1, 2, 3, 4, 4, 6, 8]
str()	Return a **str** version of *object*.	>>> str(123) '123'


Built-in Functions in Python
Modules
pi
()
sqrt
()
pow
()
pop
()
index
()
insert
()
Math
Array
Library

abs()
Return the absolute value of a number(An absolute numbers is the magnitude of real number without regard to its sign).

Syntax:
>>>abs(X)

```
# Program to illustrate usage of Abs() function.
def FindAbsolute(x):
    print("Absolute Value of Given Number is:",abs(x))

x = int(input("Enter the number:"))
FindAbsolute(x)
```

Output:
Enter the number:-90
Absolute Value of Given Number is: 90

bin()
Converts an integer number to a binary string counterpart prefixed with “0b “.

Syntax:
>>> bin(x)

```
# Program to find Binary value of a decimal.

number = int(input("Enter the Number:"))
print("Binary value of entered number
is:",bin(number))
```

Output:
Enter the Number:14
Binary value of entered number is: 0b1110

hex()
Converts an integer number to a hexadecimal lowercase string prefixed with “0x”.

Syntax:
>>>hex(x)

Note: oct() is used same as bin() and hex() to represent decimal in octal number system.

Program to find hexadecimal value of a decimal.

```
number = int(input("Enter the Number:"))
print("Hexadecimal value of entered number is:",hex(number))
```

Output:
Enter the Number:90
Hexadecimal value of entered number is: 0x5a

range()
returns an iterator from a starting point to a end point with a specific step. It is most commonly used in looping with a FOR LOOP.

Syntax:
range(start,stop[, step])

Note: Stop is a mandatory argument.
Start and Step are optional (Default value of start is 0 and step is 1).

Program to illustrate Range function.

```
for i in range(1,10,3):
    print(i)
```

Output:
1
4
7

min()
Return the smallest item in an iterable or the smallest of two or more arguments.

If one positional argument is provided, it should be an iterable. The smallest item in the iterable is returned. If two or more positional arguments are provided, the smallest of the positional arguments is returned.

Syntax:

```
>>>min(iterable)
>>>min(arg1, arg2 ,.....)
```

```
# Program to find hexadecimal value of a decimal.

number = int(input("Enter the Number:"))
print("Hexadecimal value of entered number
is:",hex(number))
```

Output:
Enter the Number:90
Hexadecimal value of entered number is: 0x5a

max()
Return the largest item in an iterable or the largest of two or more arguments.

If one positional argument is provided, it should be an iterable. The largest item in the iterable is returned. If two or more positional arguments are provided, the largest of the positional arguments is returned.

Syntax:

```
>>>max(iterable)
>>>max(arg1, arg2 ,.....)
```

```
# Program to illustrate max() function.
myList = [2,3,42,1]
print(max(myList))
print(max(33,44,22,12))
```

Output:
42
44

Operators in Python

Operators

Operators are used to performing some operation between some values of any data types. We have seven different types of Operators in Python and can perform any operation but are differentiated according to their usage.

Like when we add two numbers we use addition operators between those numbers which can be int, or float, or can be of another data type.

Arithmetic Operators in Python

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, division, etc.

Arithmetic operators can be set in between operands. In mathematical operations elements on which operation is supposed to be done are called operand and the type of operation is performed by the operator used between two operands.

Operator	Name of Operator	Example
+	Addition	Prep + Insta
–	Subtraction	Prep – Insta
*	Multiplication	Prep * Insta
/	Division	Prep / Insta
%	Modulus	Prep % Insta
**	Exponentiation	Prep ** Insta
//	Floor Division	Prep // Insta

Assignment Operators in Python

The assignment operator is used to assigning values to the variable in many different ways, we can assign values by incrementing, decrementing, or by performing any other operation and then assigning the value to the variable. We have 13 assignment operators to assign values to the variable.

Operator	Also be used as	Example
=	a = 5	a = 5
+=	a = a+5	a+=5
-=	a = a-5	a-=5
*=	a = a * 5	a*=5
/=	a = a/5	a/=5
%=	a = a%5	a%=5
//=	a = a//5	a//=5
**=	a = a **5	a**=5
&=	a = a&5	a&=5
|=	a = a|5	a|=5
^=	a = a^5	a^=5
>>=	a = a>>5	a>>=5
<<=	a = a<<5	a<<=5

Comparison Operators in Python

A comparison operator is used to comparing two or more variables or constants having some values. We can compare different variables or constants in six different ways that include, comparing if two values are equal, or if two values are not equal, and many other ways.

Operator	Name of Operator	Example
==	Equal	a == b
!=	Not equal	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

Comparison Operators in Python

A comparison operator is used to comparing two or more variables or constants having some values. We can compare different variables or constants in six different ways that include, comparing if two values are equal, or if two values are not equal, and many other ways.

Operator	Name of Operator	Example
==	Equal	a == b
!=	Not equal	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

Logical Operators in Python

The logical operator is used to perform some logical operations or combine conditional statements. Mainly logical operators are used in control statements like if, else, and elif where we can check more conditions together just by using these operators. Logical operators always return TRUE or FALSE values.

Operator	About	Example
and	TRUE if both are true	a > 5 and a > 10
or	TRUE if any one is true	a > 5 or a > 10
not	Converts TRUE to FALSE	not(TRUE)

Identity Operators in Python

Identity operators are used in Python programing language to check if the two values, variables, or constants are not only the same but also have the same memory location.
These types of operators are used to compare and check the exact match of the values between two objects.

Operator	About	Example
is	Returns TRUE if a and b both are same	a is b
is not	Return TRUE if a and b both are not same	a is not b

Membership Operators in Python

Membership operators in python are used for checking if the iterator or the given value is present in the sequence or not.
These types of operators always return values in a TRUE or FALSE format. If the value is found in the sequence then the operator will return TRUE else the operator will return FALSE.

Operator	About	Example
in	Returns TRUE if the iterator is present in sequence	a in b
not in	Returns TRUE if the iterator is not present in sequence	a not in b

Bitwise Operators in Python

These operators are used to compare the binary numbers. We can perform different operations on binary numbers such as and, or, xor, not, left shift, and right shift.
Sometimes these operators are also called bitwise-or, bitwise-and, and bitwise-xor and so on. Shift operators are used to shifting the values of the binary number to left or right.

Operator	Name	About
&	AND	Return 1 if both are TRUE or 1
|	OR	Return 1 if one of them are TRUE or 1
^	XOR	Return 1 if only one is TRUE or 1
~	NOT	Inverse
<<	Left shift	Shifting bits to the left
>>	Right shift	Shifting bits to the right

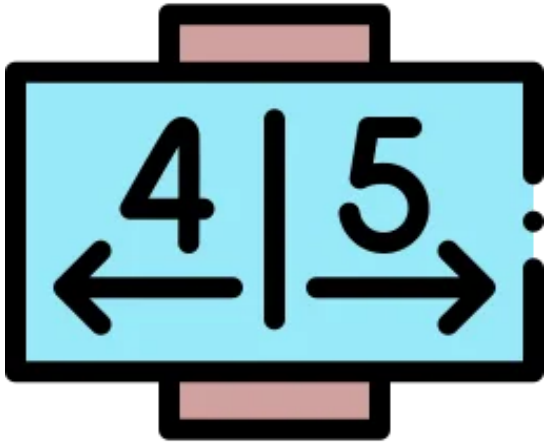
Numeric data-type in Python programming language

Data-types in python

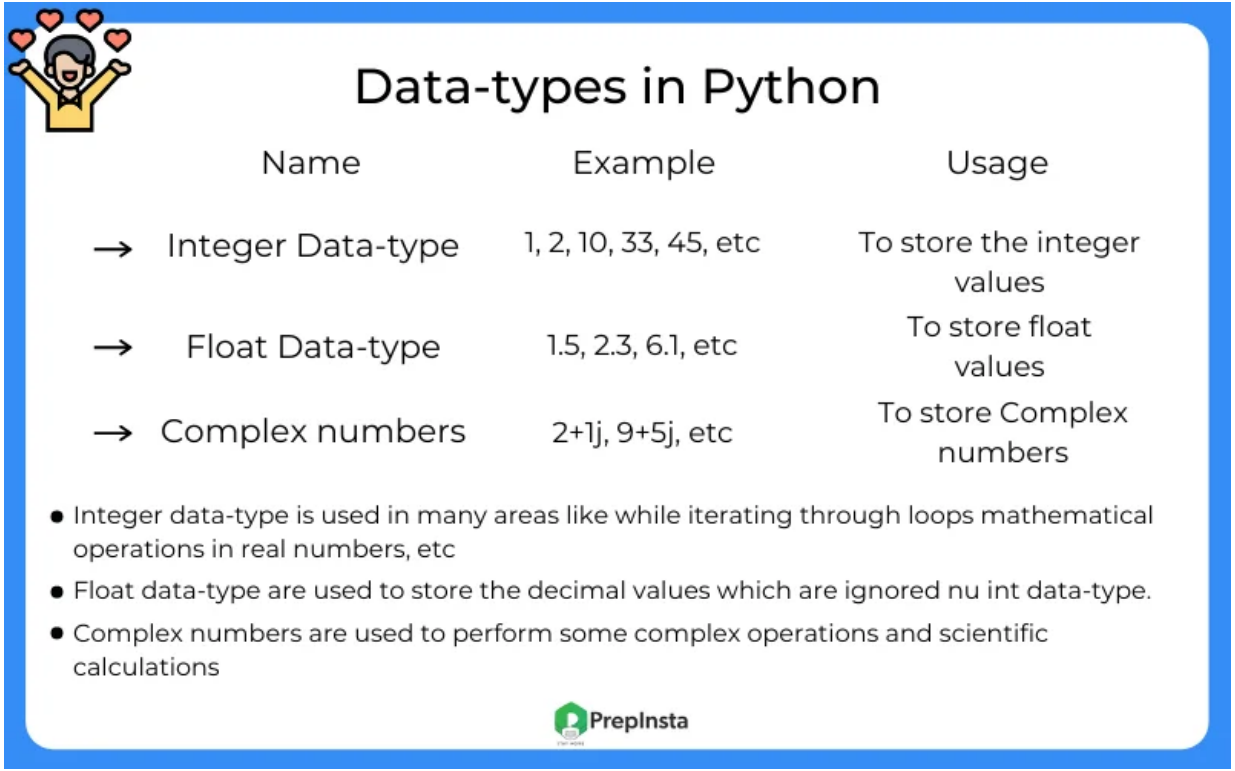
Numeric data-type in Python programming language is used to store the numeric values in any variable. Numeric data-type is used in many areas of operation. To perform some numeric operations or calculations numeric data type is used to store the values. The numeric data type is divided into three parts:

- Integer
- Float
- Complex Number

All the above sub-parts of a numeric datatype are used in different places for a different operation.



Different data-types in python



Int Data-type in Python

Integer data-types are used to perform some real number operations. Mostly int values are used in iterating through the loop and getting item indexes in the data structure.

Integers do not contain decimal point values, even if we try to save a decimal value to the integer data-type, it will automatically remove the decimal point values an stores the integer part of the value

Operations on Int data-type in Python

```
#Integer inputs
Var = int(input('Enter an integer value: '))
print(Var)
#using int in loops
#initialize an integer
i = 0
while i <= 4:
    print(i,end=" ")
    i+=1
print()

#using int to iterate through an list
#initialize a list
arr = [1,2,3,4,5]
#iterating through list
for integer in range(len(arr)):
    print(arr[integer],end=" ")
print()
#mathematical operations on integer
#initialize int variables
a = 10
b = 2
#addition
print(a+b)
#Subtraction
print(a-b)
#multiplication
print(a*b)
#division
print(a/b)
#floor division
print(a//b)
```

Output:

```
Enter an integer value: 10
0 1 2 3 4
1 2 3 4 5
12
8
20
5.0
5
```

Float data-type in Python

Float data-types are used to store the decimal numbers. Float data-types stores the numeric values written after the decimal point.

We can use format in-built function to store or print the values after decimals according to our need. Float data-types are used to calculate the exact mathematical outputs up to some decimal points.

Operations on Int data-type in Python

```
#Float inputs
Var = float(input('Enter an float value: '))
print(Var)

#mathematical operations on Float
#initialize float variables
a = 10.12
b = 2.11
#addition
print(a+b)

#Subtraction
print(a-b)

#multiplication
print(a*b)

#division
print(a/b)

#floor division
print(a//b)
```

Output:

```
Enter an float value: 10
10.0
12.229999999999999
8.01
21.353199999999998
4.796208530805687
4.0
```

Complex numbers in Python

Complex numbers are one of the Numeric data-type in Python programming language. Complex numbers are a combination of a real part and an imaginary part or imaginary value.

The real part has the real number value and the imaginary part is usually represented by “j”, represented as- 2+7j. Complex numbers are used to perform some scientific and mathematical calculations.

Operations on Int data-type in Python

```
#Complex number inputs
Var = complex(input('Enter an complex number value: '))
print(Var)

#mathematical operations on complex number
#initialize complex number variables
a = 2+4j
b = 3+6j
#addition
print(a+b)

#Subtraction
print(a-b)

#multiplication
print(a*b)

#division
print(a/b)
```

Output:

```
Enter an complex number value: 10
(10+0j)
(5+10j)
(-1-2j)
(-18+24j)
(0.6666666666666666+0j)
```

String Data-type in Python

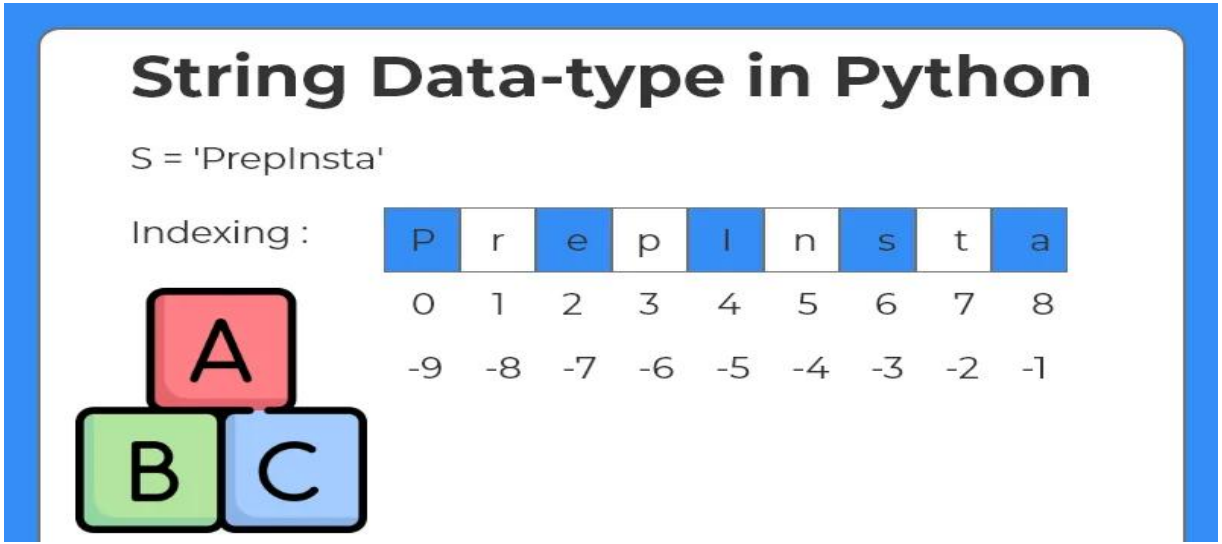
Strings

In python string is collection of characters under single or double or triple quotes. In python there is nothing like character data type , a single character is a string with length 1.

String Data-type in Python is immutable data type which means after assigning value to a variable we can't modify data from the string variable.

String data-type in Python programming language

- For string data type we use str() keyword
- We can perform many operations on String.
- We can add two string data-type using '+' operator.
- We can convert a string into a list then we can append or remove element from them.
- String supports indexing.
- There are several in built functions for strings , namely , replace(), isupper(), islower(), join(), and so on..



Example

```
#python
#creating a string

print('single quotes :')
s='Hello Prepsters welcome to PrepInsta'
print(s)

print("doubel quotes :")
s="Hello Prepsters welcome to PrepInsta"
print(s)

print('''triple quotes :''')
s='''Hello Prepsters welcome to PrepInsta'''
print(s)

print('add operation :')
s1= 'Hello Prepsters '
s2='welcom to PrepInsta'
s=s1+s2
print(s)

print('conver string into list:')
x=list(s)
print(type(x))

print('Indexing :')
s='abcdef'
print('first charcter is ;',s[0])
print('last charcter is :' ,s[-1])
```

Output:

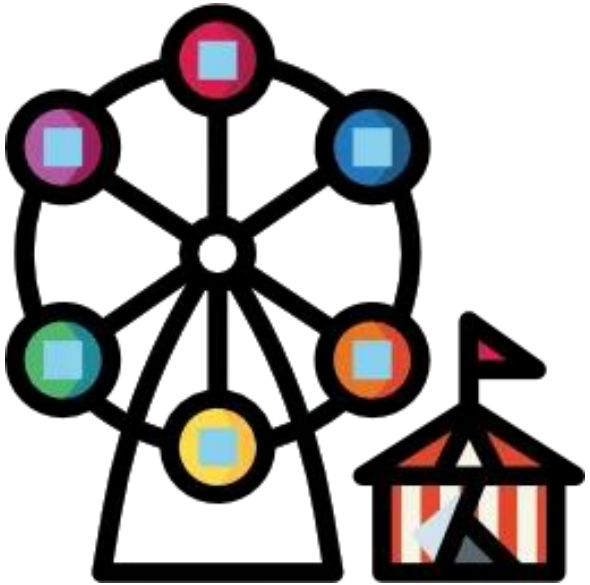
```
single quotes :
Hello Prepsters welcome to PrepInsta
doubel quotes :
Hello Prepsters welcome to PrepInsta
triple quotes :
Hello Prepsters welcome to PrepInsta
add operation :
Hello Prepsters welcom to PrepInsta
conver string into list:
<class 'list'>
Indexing :
first charcter is ; a
last charcter is : f
```

Iterator Data-types in Python programming language

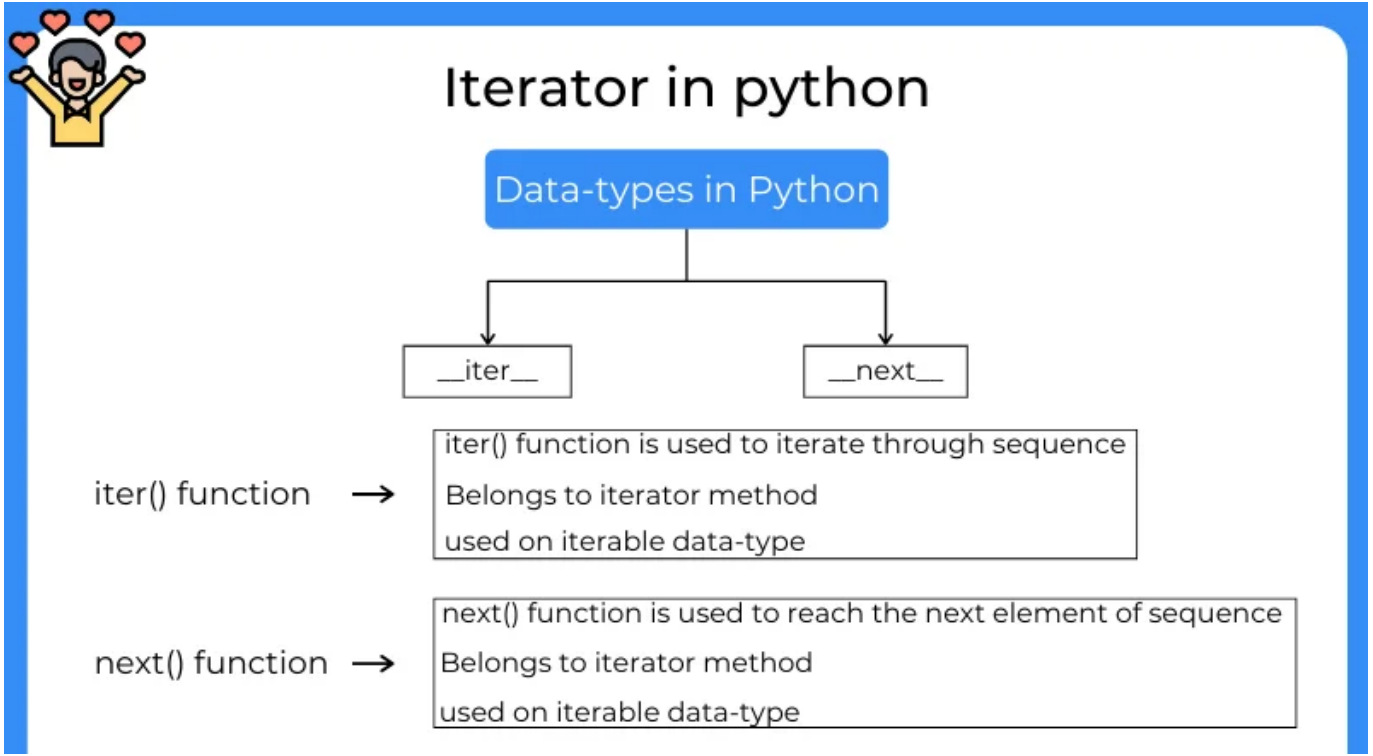
Iterator in python

Iterator Data-types in Python programming language is also a method. Iterator in While loop is a variable defined before we initialize the loop.

While the loop is used to perform operations or task where the ending is not properly defined. In while loop iterator value is manual initialized and incremented. While loop iterates through the elements of the sequence and operations are performed.



Iterator data-type



Methods of Iterator data-type

__iter__

Iterator is a method in python programming used to iterate through the elements of a sequence. iterator function comes with a function called as __iter__() function to iterate through the sequence. The iter() function takes the argument for the iterable data-type and creates it an iterable object. iter() function is used in replacement to loops. using the iter() function and using another function of iteration method we can parse the nest element

__next__

iter() function creates the object as an iterable object, but how to get the values from the sequence? Here in the frame comes the nest function of iteration method called as __next__() function. next() function is used to iterate through the elements os a sequence and checks for the next available element. If no element is found in the sequence than an exception "StopIterationError". to parse at the next element of the sequence we use the next() method.

Iterator in different control statements

Iterator in for loop

For loop is used when we want to repeat a task in the loop and ending pointy is known.
Using for loop we can perform many operations like if we want to print elements of a list or perform some task on elements of a list, etc. Iterator is the variable responsible for working of for loop. iterator iterates through the element and performs operations we want to.

Example

```
#print iterators upto a range
#take user input of range
val = int(input('Enter the ranger :'))
for i in range(val):
    print(i,end=' ')
print()

#iterating through list data-type
lis = [1,2,3,4,5,6]
for i in range(len(lis)):
    print(lis[i],end=' ')
print()

#iterating through string data-type
string = 'PrepInsta'
for i in string:
   print(i,end=' ')
print()

#iterating through dictionary
dictio = {1:'Divyanshu', 2:'875597', 3:'CS'}
for i in dictio.keys():
    print(i,dictio[i])
```

Output:

```
Enter the ranger :5
0 1 2 3 4
1 2 3 4 5 6
P r e p I n s t a
1 Divyanshu
2 875597
3 CS
```

Iterator in while loop

Iterator in While loop is a variable defined before we initialize the loop.

While the loop is used to perform operations or task where the ending is not properly defined.

In while loop iterator value is manual initialized and incremented. While loop iterates through the elements of the sequence and operations are performed.

Example

```
#print iterators upto a range
#take user input of range
val = int(input('Enter the ranger :'))
i = 0
while i < val:
    print(i)
    i+=1
#iterating throug list data-type
lis = [1,2,3,4,5,6]
i = 0
while i < len(lis)
    print(lis[i],end=' ')
    i==1
print()

#itaratring through string data-type
string = 'PrepInsta'
i = 0
while i < len(string):
    print(string[i],end=' ')
    i+=1
print()
```

Output:

```
Enter the ranger :5
0 1 2 3 4
1 2 3 4 5 6
P r e p I n s t a
```

__iter__ method

```
#iter function using while loop
#initialize a variable
variable = '12345'
#pass the variable to iter function
iter_value = iter(variable)
while True:
    try:
        #iterating to the next elements of iterable
        element = next(iter_value)
        print(element,end=' ')
    #if it reaches the limit of length of an iterator.
    except StopIteration:
    break
print()

#iter function using while loop
#initialize a variable
variable = 'PrepInsta'
#pass the variable to iter function
iter_values = iter(variable)
for i in range(len(variable)):
    #iterating to the next elements of iterable
element = next(iter_values)
print(element,end=' ')
```

Output:

```
1 2 3 4 5
P r e p I n s t a
```

__next__ method

```
#initialize an list
arr = [1410, 'DJ', 19.97]

#pass the list to iter function
next_arr = iter(arr)

#using next function iterate over the next element
element = next(next_arr)
print(element)

element = next(next_arr)
print(element)

element = next(next_arr)
print(element)

#check for more result
try:
element = next(next_arr)
print(element)
#handle the exception
except StopIteration:
print('No more elements')

#next function using while loop
#initialize a variable
variable = '12345'
#pass the variable to iter function
iter_value = iter(variable)
while True:
   try:
      #iterating to the next elements of iterable
      element = next(iter_value)
      print(element,end=' ')
   #if it reaches the limit of length of an iterator.
   exceptan StopIteration:
      break
print()
```

Output:

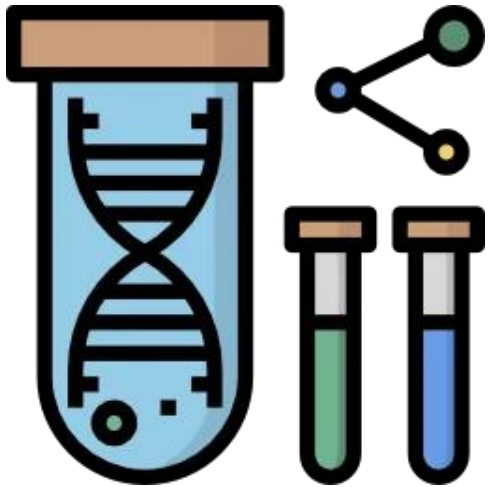
```
1410
DJ
19.97
No more elements
1 2 3 4 5
```

Sequence Data Types in Python programming language

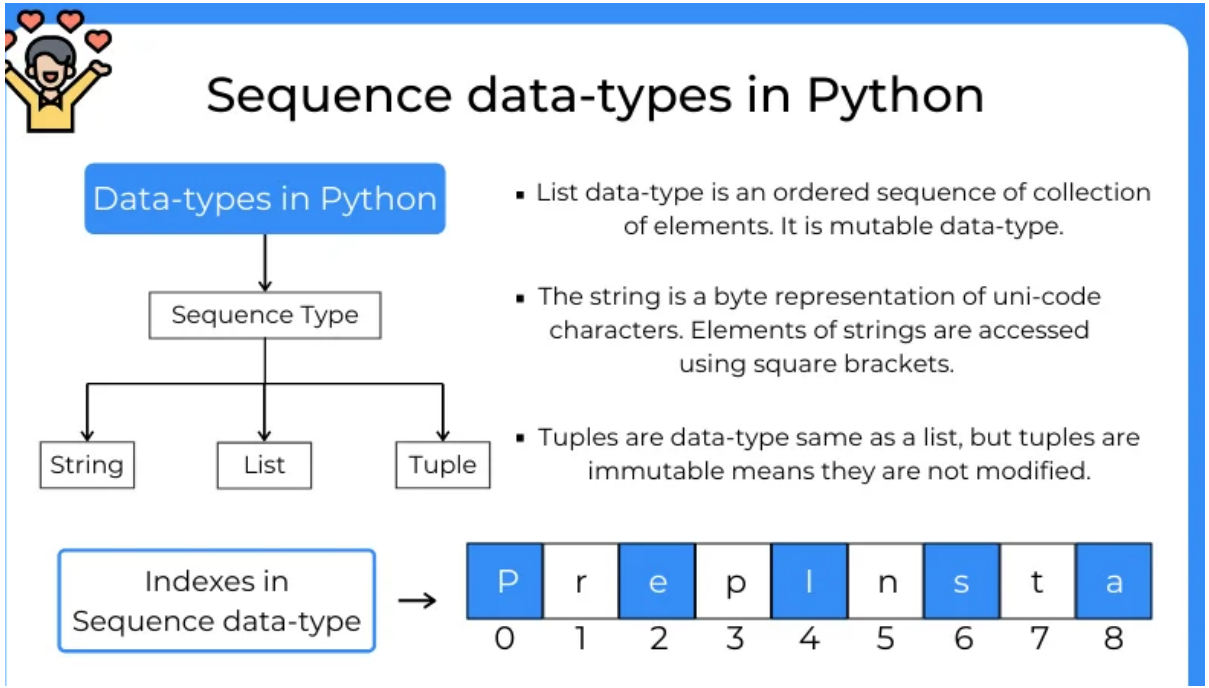
Sequence data-type

Sequence Data Types in Python programming language is used to store data in containers. List, Tuple and String are the different types of containers used to store the data.

Lists can store the data of any data type and are mutable, Strings can store data of the str type and are immutable. Tuples can also store any type of value and are immutable data-types.



Different Sequential data-types in Python



List in Python

List data-type belongs to the sequential data-type class. The list is the only data-type under sequential which are mutable. It can store values or elements of any data-type. We can change and perform many operations on the list and like

- append
- remove
- reverse
- sorted
- Extend

etc. We have many more built-in functions available for manipulation in lists. Let's have a look at some important operations on the list.

Operations on List

```
# initialize an empty list
arr = [1, 4, 2, 5, 3]

# sorting the list
arr = sorted(arr)
print(arr)

# minimum element of the list
minimum = min(arr)
maximum = max(arr)
print(minimum)
print(maximum)

# reverse the list
print(arr[::-1])
```

Output:

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

String in Python

String data-types are used to store the string values. Strings are immutable and so we cannot manipulate the elements in the string. Strings although have many built-in functions and we a perform many operations on the string. Some built-in functions available for strings are:

- count
- isupper
- islower
- join
- Split

etc. We have many other functions for string data-type. Let’s have a look at some operations on strings.

Operations on String

```
# initialize a string
String = 'PrepInsta'

# counting number of times a is available in String
val = String.count('a')
print(val)

# converting string to uppercase
String = String.upper()
print(String)

# converting string to lowercase
String = String.lower()
print(String)

# reverse the String
print(String[::-1])

# join in String
String = '-'.join(String)
print(String)

# split in string
String = String.split('-')
print(String)
```

Output:

```
1
PREPINSTA
prepinsta
atsniperp
p-r-e-p-i-n-s-t-a
['p', 'r', 'e', 'p', 'i', 'n', 's', 't', 'a']
```

Tuple in Python

Tuples also belong to the class of sequence data-type. There are almost the same as the list in python but have the property of immutability. We cannot manipulate the elements of the tuple but we can perform many operations on tuples:

- count
- index
- len
- Type

etc. Let’s have a look at some operations on tuples.

Operations on Tuple

```
# initialize a tuple
Tu = tuple([1, 'String', 1.25, 4, 6])
print(Tu)

# counting elements in tuple
val = Tu.count(1)
print(val)

# finding index of an element in tuple
ele = Tu.index(1.25)
print(ele)

# finding type of sequence
print(type(Tu))

# finding length of tuple
length = len(Tu)
print(length)
```

Output:

```
(1, 'String', 1.25, 4, 6)
1
2
<class 'tuple'>
5
```

Block and statement in Python

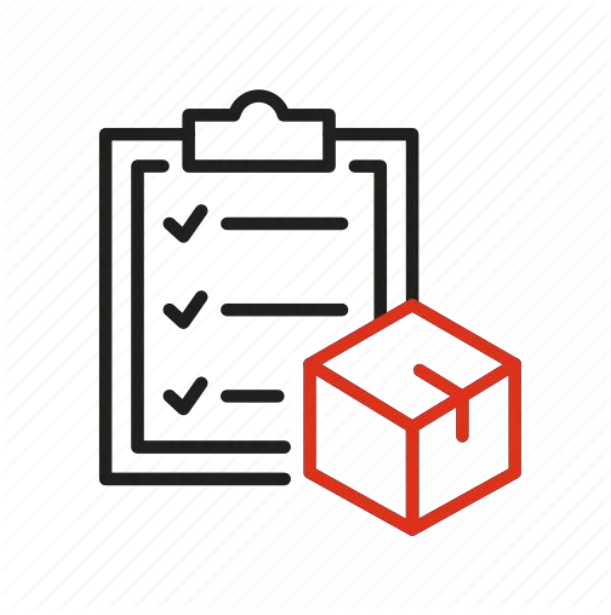
Block and Statement

What is a Block and statement in python ?
A block is a piece of Python program text that is executed as a unit. The following are blocks: a module, a function body, and a class definition. Each command typed interactively is a block.

Statement : Instructions that a Python interpreter can execute are called statements.

Example :

- def add(a,b):
  #Block
- P=’PrepInsta’ #statement



Block and statement in Python

Block :

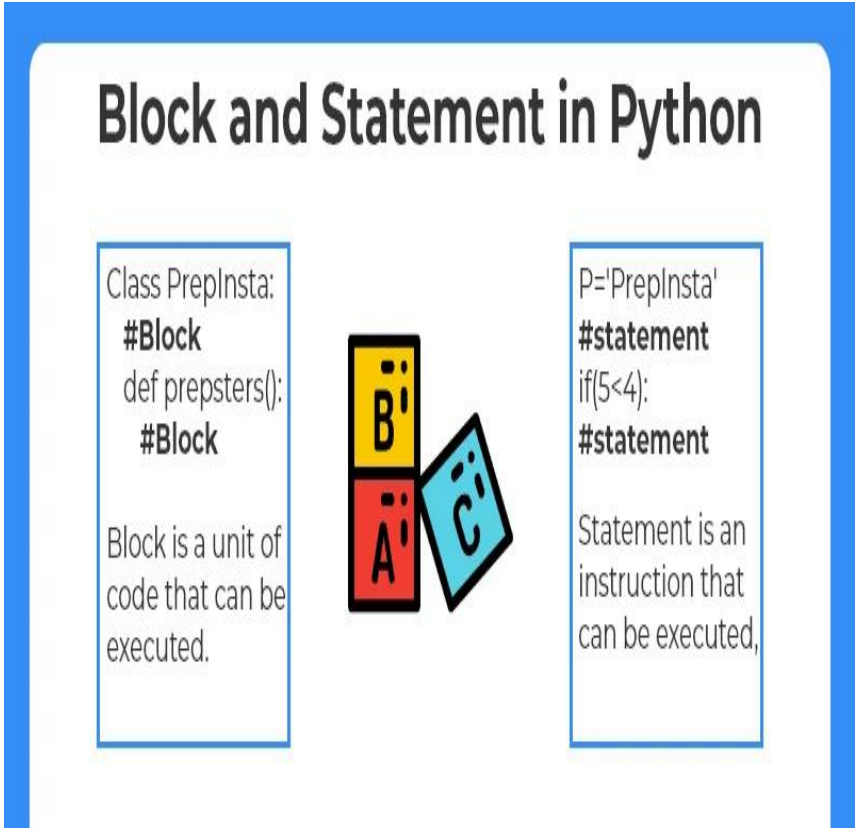
- A Python program is constructed from code blocks.
- A script file (a file given as standard input to the interpreter or specified as a command line argument to the interpreter) is a code block.

Statement :

- There are different types of statements in the Python programming language like Assignment statement, Conditional statement, Looping statements etc.

  P=’PrepInsta’ #assignment statement

- Multi-Line Statements: In Python we can make a statement extend over multiple lines using braces {}, parentheses (), square brackets [], semi-colon (;), and continuation character slash (\).



Implementation of Block and Statement in Python

```
#Block
def add(A,b):
   #block
Class Student:
   #block
   def(self,name,roll):
      #block


#Statements
#Continuation Character (\):
s = 1 + 41 + 48 + \
   4 + 51 + 6 + \
  50 + 10

#parentheses () :
n = (17 * 5 * 4 + 8 )

#square brackets [] :
cars = ['BMW',
        'THAR',
        'FERARI']

#braces {} :
x = {1 + 2 + 3 + 4 + 5 + 6 +
    7 + 8 + 9}

semicolons(;) :
a= 2; b = 3; c = 4
```

Indentation in Python

Indentation
Indentation is a very important concept of Python because without proper indentation the Python code will end up with IndentationError and the code will not get compiled.

Python indentation is a way of telling a Python interpreter that the group of statements belongs to a particular block of code. A block is a combination of all these statements. Block can be regarded as the grouping of statements for a specific purpose

Example:

- def add():

#one tab space is preferd as indention generally.

Indentation in Python

Indentation :

- It simply means adding whitespace before statement.
- 1 Tab space is considered as one indentation.
- 1 Tab space is equal to 4 white space.
- Genrally , after every ':' we use Indentation.

Example 1

```
#indendation
p='PrepInsta'
if(p=='PrepInsta'):
    print('Hello Prepsters we are here to help you.')
```

Output:

```
Hello Prepsters we are here to help you.
```

```
 #without indentation
p='PrepInsta'

if(p=='PrepInsta'):
print('Hello Prepsters we are here to help you.')
```

Output:

```
 File "main.py", line 4

    print('Hello Prepsters we are here to help you.')
      ^
IndentationError: expected an indented block
```

Comments in Python

Commenting in Python Code

Comments in Python Code means removing some part of code from the operation without deleting it from the text area.

Comments are used in every programming language to write some none coding information about what operation is performed in the below or a specific line. Let’s learn more on Comment in Python programming.

What does the comment do in Python?

Comments are used in every programming language to remove the operability of the line of code. We can comment single line and even multiple lines in code. We use some different symbols to eliminate the operability of the piece of code. Different methods to comment on the code are:-

1. #:- Used to comment on the single line of code in Python.
2. ”’.....”’:- Used to comment on multiple lines in Python
3. “””.....”””:- Also Used to comment on multiple lines

We can use any of the type mentioned above to comment on the python code. Not only the python code Comments are also used by programmers to make their codes readable. Commenting the code gives a clear explanation about what operation programmer want to perform in the coming line.

Using Comment lines can help you to save more time while implementing operations. For example, Initially, if we performed an operation to a program like sorting an array in ascending order. But now you want to run the same code with a sorted array in descending order. So you can comment the code for the time of not using it and again remove the comment when you want to run it in your programme.

Comments in Python

Their are two ways to comment in python

Using '#' Special Character

- Hashtag is used to comment on single line

Using '''.....'''Special Character

- Triple inverted commas used to comment on multiple line

Example

Silgle line comment

```
#initialize a string
arr = 'PrepInsta'
#using step argument without start
and stop
#this will print the string as it is
print(arr)
```

Multiple line comment

```
#initialize a string
arr = 'PrepInsta'
'''using step argument without start
and stop
this will print the string as it is'''
print(arr)
```

Advantages of using Comments in Code

Some advantages of usding comments in your code are as follows:

- Makes the code readable
- Make your code easy to understand
- Saperate your functions and methords in a better way

Code for Comments in Python

```
#initialize a string
arr = 'PrepInsta'
#using step argument without start and stop
#this will print the string as it is                #commented the lines using single comment format
print(arr)
'''This will print the string in reverse order
starting from the index 2 to ending point 6'''          #Commented the lines using multiple line comment format
ar = arr[2:7]
print(ar[::-1])

"""This will print the string elements with even index
Starting from 2 to 7"""                          #Commented the lines using multiple line comment format
print(arr[2:8:2])
```

Output:

```
PrepInsta
snIpe
eIs
```

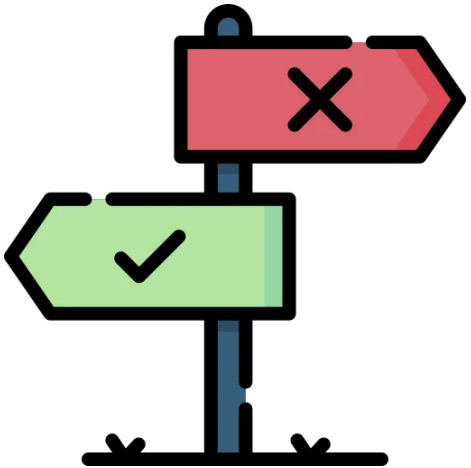
Bool() in python

Bool
Bool() is an built-in function in python. It returns the boolean value of a speciified object i.e is True or False using standard truth testing procedure .

Syntax :
- bool(x)

Example :

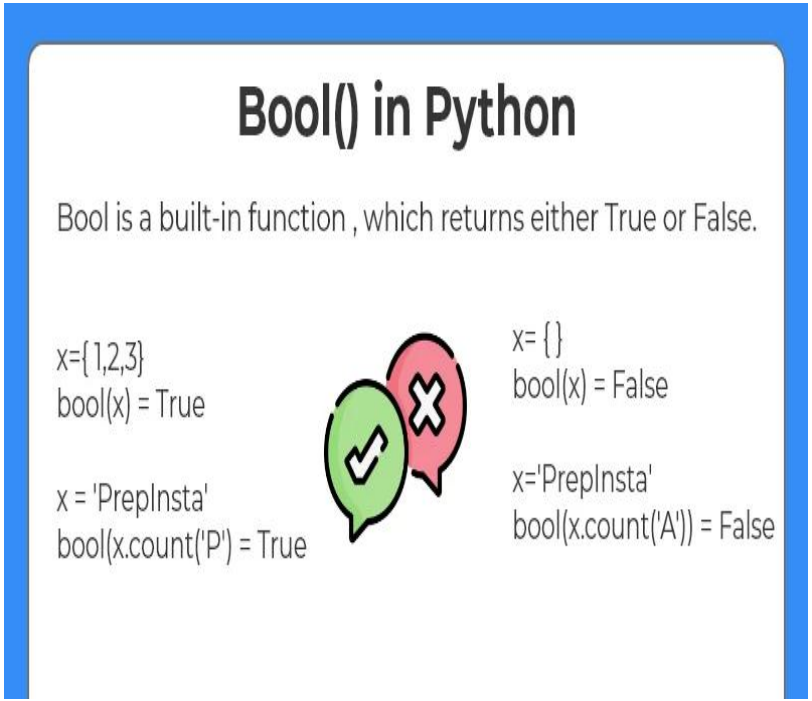
- x=bool(1) #return the boolean value of 1.



Bool() in python

Bool :

- It generally takes one value as parameter.
- If no values passed it will return default value which is False.
- If value passed is true it will True else False.
- In some of cases it will always return False
    - If None is passed.
    - If an empty list , set or any sequence is passed .
    - If zero is passed .
    - If and function inside bool function returns zero i.e len() or count().



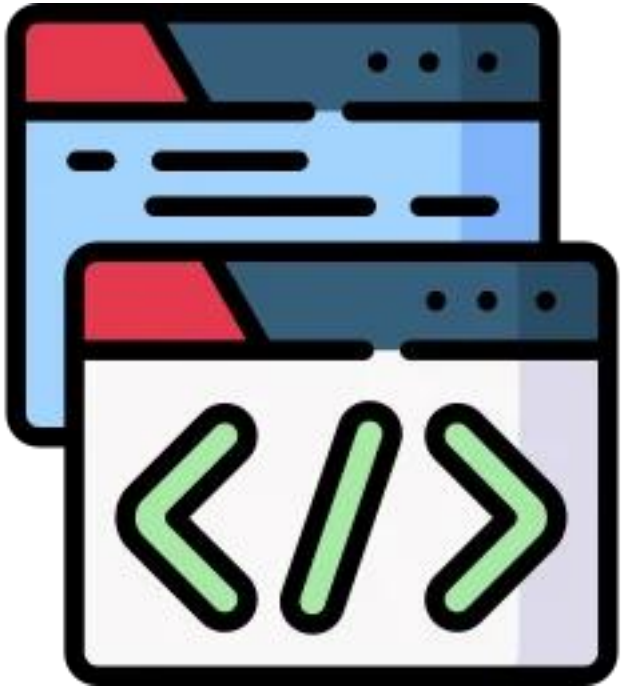
```
#Python program
# built-in method bool()

# Returns True as x is True
x = True
print(bool(x))

# Returns False as x is False
x = False
print(bool(x))

# Returns False as x is not equal to y
x = 4
y = 14
print(bool(x==y))

# Returns False as x is None
x = None
print(bool(x))

# Returns False as x is an empty sequence
x = []
print(bool(x))

# Returns False as x is an empty mapping
x = {}
print(bool(x))

# Returns False as x is 0
x = 0.0
print(bool(x))

# Returns True as x is a non empty string
x = 'Prepinsta'
print(bool(x))

#Returns True as length of x > 1
x=(1,2)
print(bool(len(x)))
```

Output:

```
True
False
False
False
False
False
False
True
True
```

Slicing in Python

- Slice() function in python
- Slicing in python
- Slicing with negative numbers
- Using step in a slicing
- String operators in python
- String replacement fields in python
- F-string in python
- String interpolation in python
- Truth-value testing in python



slice() Function in Python programming language

slice() Function

Slice() Function in Python programming language is a built-in function used to iterate through the sequences. Slice function takes three arguments 1st is starting point 2nd is ending point and 3rd for the order to print the element. Sequence data-type is list, tuple, the string is the data-type where we can perform slice operation using slice function.

Slicing in Python

R	o	y	a	l	p	a	s	s
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

arr = slice(2)

R	o

- Slicing with 1 argument

arr = slice(1,4)

o	y	a

- Slicing with 2 arguments

arr = slice(-1)

R	o	y	a	l	p	a	s

- Slicing with 3rd negative arguments

arr = slice(1,5,1)

o	y	a	l

- Slicing with all 3 arguments

slice() function for list

The list is a sequential data-type and is mutable. we can perform operations on the list using the slice built-in function.

Slice function in the list takes 3 arguments where one of them is a mandatory field and two are optional. Let's have a look at some operations on the list using builtin- slice(0 functions.

Python code for slice function on list

```
arr = ['P','r','e','p','l','n','s','t','a']

#Slicing in list with 2 arguments
a = slice(1,4)
print(arr[a])

#slicing in list with 1 argument
a = slice(2)
print(arr[a])

a = slice(-1)
print(arr[a])

#Slicing in list with 3 arguments
a = slice(1,5,1)
print(arr[a])
```

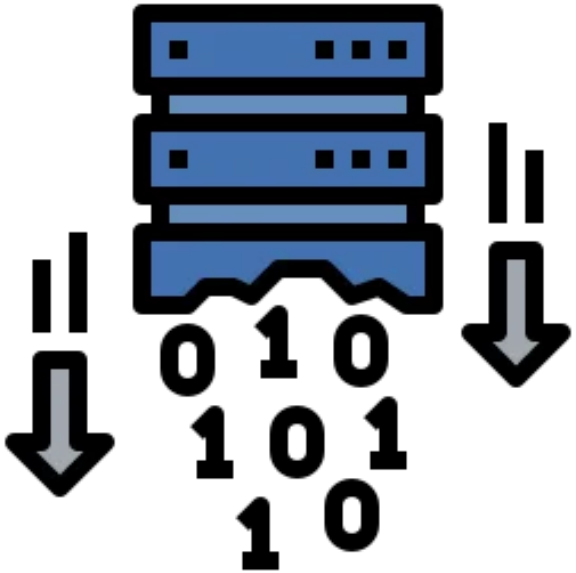
Output:

```
['r', 'e', 'p']
['P', 'r']
['P', 'r', 'e', 'p', 'l', 'n', 's', 't']
['r', 'e', 'p', 'l']
```

slice() function for Tuple

A tuple is same as list data-type. But the major difference between list and tuple is that tuple is an immutable data-type.

Although we can perform some operations on the tuple. Since built-in function takes the arguments and represents the tuple from a starting point to endpoint.



Python code for slice() function on tuple

```
arr = tuple(['P','r','e','p','l','n','s','t','a'])

#Slicing in list with 2 arguments
a = slice(1,4)
print(arr[a])

#slicing in list with 1 argument
a = slice(2)
print(arr[a])

a = slice(-1)
print(arr[a])

#Slicing in list with 3 arguments
a = slice(1,5,1)
print(arr[a])
```

Output:

```
['r', 'e', 'p']
['P', 'r']
['P', 'r', 'e', 'p', 'l', 'n', 's', 't']
['r', 'e', 'p', 'l']
```

slice() in Strings

String data-type is an immutable data-type in python. But we can perform many operations on strings than on tuples.

Strings are iterable and so we can use built-in function Slice() to print of use the sub-part of string according to our need. Lets have a look at some representations of string using slice function.

Python code for slice() function on String

```
arr = 'PrepInsta'

#Slicing in string with 2 arguments
a = slice(1,4)
print(arr[a])

#slicing in string with 1 argument
a = slice(2)
print(arr[a])

a = slice(-1)
print(arr[a])

#Slicing in string with 3 arguments
a = slice(1,5,1)
print(arr[a])
```

Output:

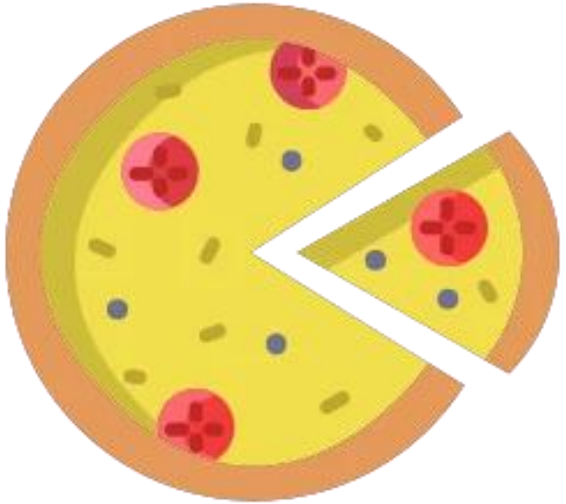
```
rep
Pr
PrepInst
repl
```

Slicing in Python programming language

Slicing

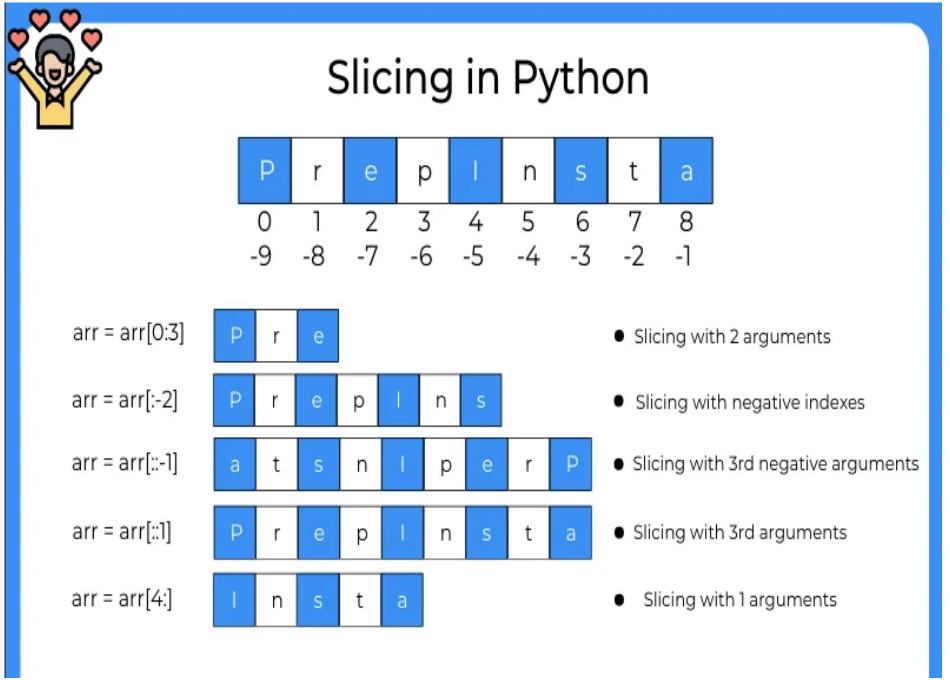
Slicing in Python programming language is a method of extracting elements of sequence from a starting point t ending point. Slicing can be performed on any of the data-type like list, tuple, string, etc.

Slicing takes three arguments where any one is mandatory and rests two are optional. We generally pass indexes of the elements of a sequence to get the desired output.



Advantages of Slicing in Python programming

- An efficient way of iterating through the sequence.
- Can reverse any sequence easily.
- Formatting of sequence elements from a starting point to the ending point in an efficient way.



Slicing in different data-type

Slicing in List

The list is an iterable data-type and hence we can slice the list. The list is the same as an array in another programming language. By using slicing we can traverse through the whole list, we can reverse it and print it from a specific start index to ending index.

Python code for slice() function on String

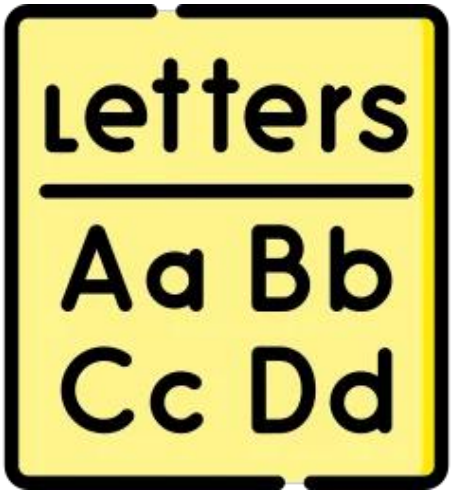
```
arr = ['string', 1, 9, 5.12, 6]

#Slicing in list with 2 arguments
print(arr[1:4])

#slicing in list with 1 argument
print(arr[2:])
print(arr[:3])
print(arr[::1])

#Slicing in list with 3 arguments
print(arr[1:3:1])
print(arr[1:3:-1])

#reversing an string by Slicing
print(arr[::-1])
```

Output:

```
[1, 9, 5.12]
[9, 5.12, 6]
['string', 1, 9]
['string', 1, 9, 5.12, 6]
[1, 9]
[]
[6, 5.12, 9, 1, 'string']
```

Slicing in String

The string is an immutable data-type and so we cannot modify the string. But as the string is iterable we can access elements, reverse the string or print the elements from a starting point to an ending point. Let's us see slicing in the string.



Python code for slicing in String

```
arr = 'PrepInsta'

#Slicing in list with 2 arguments
print(arr[1:4])

#slicing in list with 1 argument
print(arr[2:])
print(arr[:3])
print(arr[::1])

#Slicing in list with 3 arguments
print(arr[1:3:1])

#reversing an string by Slicing
print(arr[::-1])
```

Output:

```
rep
epInsta
Pre
PrepInsta
re
atsnIperP
```

Slicing in Tuple

The tuple is an immutable data-type of sequential data-type family. Tuples are iterable and we can perform operations on them.

The tuple is a sequential data-type same as a list but the only difference is lists are mutable. Slicing is a good method to iterate through tuple and use tuple elements to perform operations.

Python code for slicing in tuple

```
arr = tuple('PrepInsta')

#Slicing in list with 2 arguments
print(arr[1:4])

#slicing in list with 1 argument
print(arr[2:])
print(arr[:3])
print(arr[::1])

#Slicing in list with 3 arguments
print(arr[1:3:1])

#reversing an string by Slicing
print(arr[::-1])
```

Output:

```
('r', 'e', 'p')
('e', 'p', 'I', 'n', 's', 't', 'a')
('P', 'r', 'e')
('P', 'r', 'e', 'p', 'I', 'n', 's', 't', 'a')
('r', 'e')
('a', 't', 's', 'n', 'I', 'p', 'e', 'r', 'P')
```

Slicing with Negative Numbers in Python

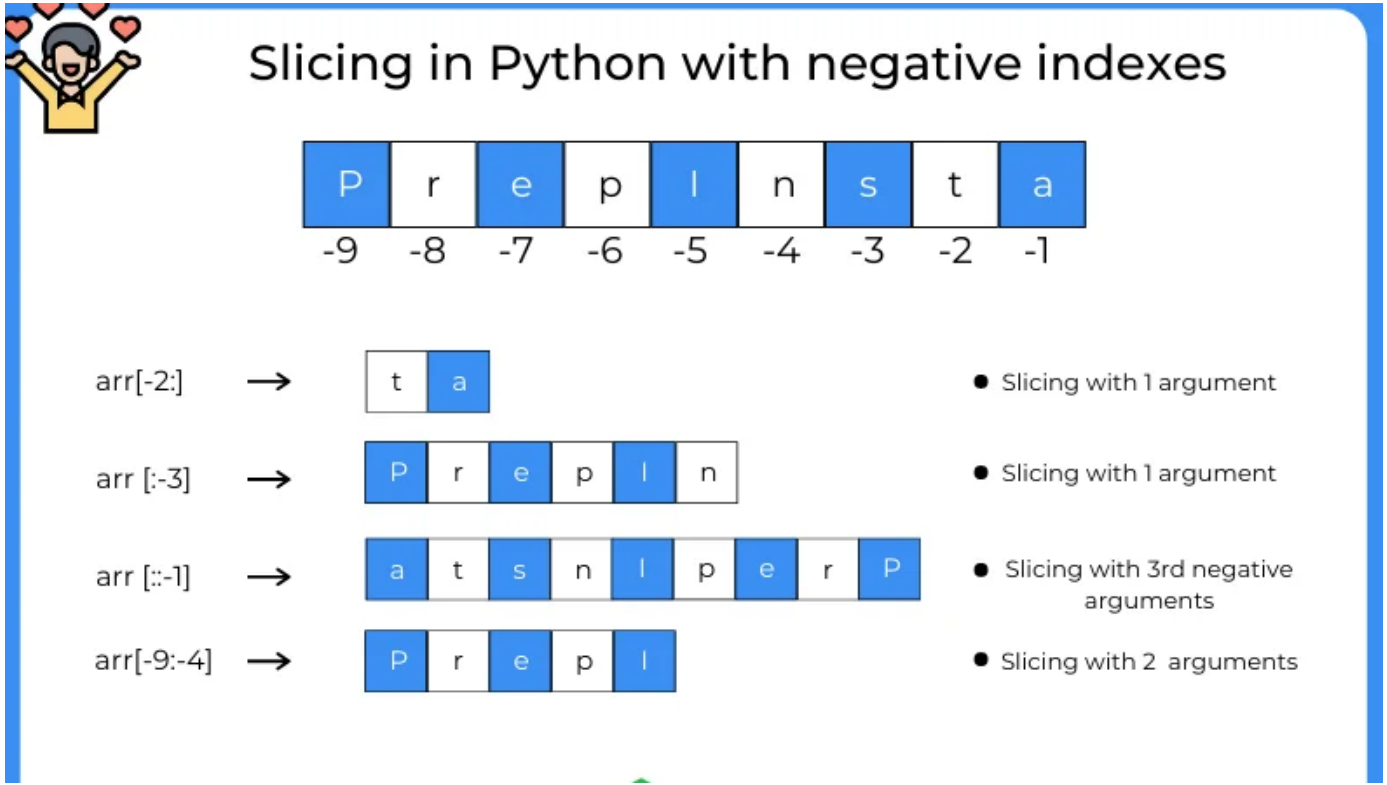
Slicing in Python with negative indexes

Slicing with Negative Numbers in Python is a subpart of slicing in python. Slicing is used to iterate through the sequential or iterable data-type in the python programming language. In these Section, we will look at the slicing on some iterable or sequential data-types like:

- List
- String
- Tuple

We will use negative indexes to slicer the elements of the data-types.

Slicing with negative numbers in Python



Slicing in String with negative indexes

The string is a data-type belongs to sequence data-type category. Slicing in the string is used to iterate through the elements.

We have negative as well as negative indexes of any iterable data-type. In the below python code we will be iterating through negative indexes of the string elements.

```
#Initialize the String
String = 'PrepInsta'

#Slicing using 1 negative index
arr = String[-2:]
print(arr)

#Slicing using 1 negative index
arr = String[:-3]
print(arr)

#Slicing using 3rd negative index
arr = String[::-1]
print(arr)

#Slicing using 2 negative indexes
arr = String[-9:-4]
print(arr)
```

Output:

```
ta
PrepIn
atsnIperP
PrepI
```

Slicing in List with negative indexes

The list is a data-type in Python programming language. Slicing is an iterable data-type and so we can perform some slicing operations on the string. Slicing is done using indexes of the iterable. Here we will iterate through the list using the negative indexes.

```
#Initialize the String
String = ['P', 'r', 'e', 'p', 'l', 'n', 's', 't', 'a']

#Slicing using 1 negative index
arr = String[-2:]
print(arr)

#Slicing using 1 negative index
arr = String[:-3]
print(arr)

#Slicing using 3rd negative index
arr = String[::-1]
print(arr)

#Slicing using 2 negative indexes
arr = String[-9:-4]
print(arr)
```

Output:

```
['t', 'a']
['P', 'r', 'e', 'p', 'l', 'n']
['a', 't', 's', 'n', 'l', 'p', 'e', 'r', 'P']
['P', 'r', 'e', 'p', 'l']
```

Slicing in Tuple with negative indexes

Tuple data type is sequential data-type in python programming. A tuple is an immutable data-type. But we can perform many operations on the tuple. As it is an iterable data-type we can perform some slicing functions on the tuple. Let's have a look at the python codes for slicing on the tuple.

```
#Initialize the String
String = tuple(['P', 'r', 'e', 'p', 'l', 'n', 's', 't', 'a'])

#Slicing using 1 negative index
arr = String[-2:]
print(arr)

#Slicing using 1 negative index
arr = String[:-3]
print(arr)

#Slicing using 3rd negative index
arr = String[::-1]
print(arr)

#Slicing using 2 negative indexes
arr = String[-9:-4]
print(arr)
```

Output:

```
('t', 'a')
('P', 'r', 'e', 'p', 'l', 'n')
('a', 't', 's', 'n', 'l', 'p', 'e', 'r', 'P')
('P', 'r', 'e', 'p', 'l')
```

Using Step in a Slice

Step in Slice

We can perform many operations on a list object Using Step in a Slice function in Python.

Slice function is operable on list structures and it takes exactly 3 arguments where the 3rd argument is the step perimeter. Slicing on the string can be performed with or without using this argument.

How to use Step in Slice?

Slice function is no something like a built-in function it is just an operation performed on list structures or on the structures which are iterable. Some basic operations like printing a substring from a known starting point to an ending point, or printing the whole string starting after the specific index etc. It exactly takes Three arguments as follows:

1. Start:- It is the first argument that we pass when we use the slice function. It takes the index from where you want to start printing or iterating the list or an iterable structure.
2. Stop:- It is the second argument that we need to pass when we use the slice function over any list or iterable structure. It takes the ending point or the point up to where you want to access the element.
3. Step:- It is the third argument that we pass while iterating through any list or iterable structure, which is also an optional argument. We will be understanding more about the argument in detail below.

Slicing in Python with Step argument

Indexes in Sequence data-type →

P	r	e	p	l	n	s	t	a
0	1	2	3	4	5	6	7	8

Examples of using step argument

arr [::-1] →

a	t	s	n	l	p	e	r	P

arr [::1] →

P	r	e	p	l	n	s	t	a

arr [::2] →

P	e	l	s	a

Advantages of using Step argument in Slice function

Every operation we perform on some structure has some advantages Lets have a look at Some of the advantages of using step function in Python programming language.

1. Easy to iterate over iterable structures.
2. Reduces the complexity to perform some complex operations like reverse the list or string.
3. Makes it easy to perform operations on elements in any sequence without using looping statements.
4. It is an optional argument, so compiler never throws an error if we forget to pass the value.

Python Code to use Step argument in Slicing

```
#initialize a string
arr = 'PrepInsta'
#using step argument without start and stop
#this will print the string as it is
print(arr[::1])
#This will print the string in reverse order
print(arr[::-1])

#This will print the string elements with even index
print(arr[::2])

#This will print the string with odd indexes
print(arr[::3])

#This will print the array element in reverse order with even indexes
print(arr[::-2])
```

Output:

```
PrepInsta
atsnIperP
PeIsa
Pps
asIeP
```

In the above code snippet, we have seen the usage of step argument without the rest two start and stop perimeter, Now let's use step argument with all the perimeter.

Python Code to use Step argument in Slicing

```
#initialize a string
arr = 'PrepInsta'
#using step argument without start and stop
#this will print the string as it is
print(arr[0:9:1])
#This will print the string in reverse order
#startring from the index 2 to ending point 6
ar = arr[2:7]
print(ar[::-1])

#This will print the string elements with even index
#Starting from 2 to 7
print(arr[2:8:2])

#This will print the string with odd indexes
#Starting from index 1 to 6
print(arr[1:7:3])

#This will print the array element in reverse order with even indexes
ar = arr[0:9]
print(ar[::-2])
```

Output:

```
PrepInsta
snIpe
eIs
rI
asIeP
```

Python – String Operators

String Operators –
There are few operators in python which can be used with numeric operands as well as strings
The “+” operator
The “*” operator
Besides this two there is one membership operator which can be used on all sequences like string, list etc.

The “in” operator
Let’s discuss what this operators do in detail:

The “+” operator
Concatenates strings
Returns a string consisting of the operands joined together

Example
The below example shows how " + " operator works with the help of Python shell.

```
>>> str1 = "Prep"
>>> str2 = "Insta"
>>> str1+str2
'PrepInsta'
>>> print("Hello"+" "+"team"+"!")
"Hello team!
```

The “*” operator
•**Creates multiple copies of strings**
- Returns a string consisting of the n concatenated copies of a string

Example
The below example shows how " * " operator works with the help of Python shell.

```
>>> str1 = "Prep"
>>> n = 3
>>> str1 * n
'PrepPrepPrep'
>>> (str1 + " " )*n
'Prep Prep Prep '
>>> str1*-2
''  # if negative integer is used with * operator then an empty string('')
') is returned
>>> str1*3
'PrepPrepPrep'
```

The “in” operator

- **A membership operator that can be used with strings**
- returns True if the first operand is contained within the second
- returns False otherwise
- also can be used as not in

Example

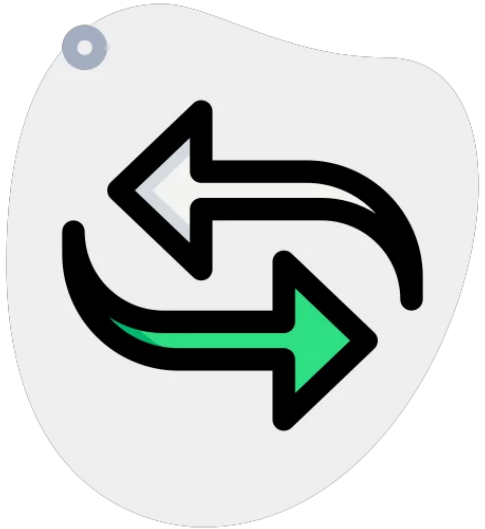
The below example shows how " in " operator works with the help of Python shell.

```
>>> str1 = "Let's start preparing!"
>>> "!" in str1
True
>>> "?" in str1
False
>>> "?" not in str1
True
```

String Replacement Fields in Python

Python String Replacement Fields

Python is a way easier to write code as we compare it with other programming languages. Like all other languages python also have a string modulo operator which is generally used to format string data. But python also offers some string replacement fields for ease of doing string formatting that's why Python is one of the favourite programming languages.



Here we will learn:

- The string.format() method

The Python String .format() Method
Syntax

<template>.format(<positional_argument(s)>, <keyword_argument(s)>)

Note that it is a technique, not an operator. The < template > method, which is a string containing the replacement fields, is called. The method specifies < positional arguments > and < keyword arguments > values that are inserted into < template > instead of the replacement fields. The resulting formatted string is the return value of the method.

Replacement fields are enclosed in curly braces ({}) in the < template > string.A literal text that is copied directly from the template to the output is anything not contained in curly braces. If you need to include, in the template string, a literal curly bracket character, such as {or}, then you can escape this character by doubling:

```
>>> '{{ {0} }}'.format('Hello')
'{ Hello }'
```

1. Positional Arguments:

Positional arguments are inserted instead of numbered replacement fields into the template. Like list indexing, replacement field numbering is zero-based. The number of the first positional argument is 0, the number of the second is 1, and so on:

```
>>> '{0}/{1}/{2}'.format( 'indb',  'ndls',  '007')
 'indb/ndls/007’
```

Note that replacement fields don’t have to appear in numerical order in the template. In any order, they can be specified, and they can appear more than once:

```
>>>  '{2}.{1}.{0}/{0}{0}.{1}{1}.{2}{2}'.format('indb', 'ndls', '007')
 '007.ndls.indb/indbindb.ndlsndls.007007'
```

The numbers in the substitute fields can be omitted, in which case the interpreter assumes a sequential order. This is known as automatic numbering of a field:

```
>>> '{}/{}/{}'.format('indb', 'ndls', '007')
 'foo/bar/baz'
```

Note

These two techniques can't be combined in a single operation.

2. Keyword Arguments:

Keyword arguments are inserted into the template string in place of keyword replacement fields with the same name:

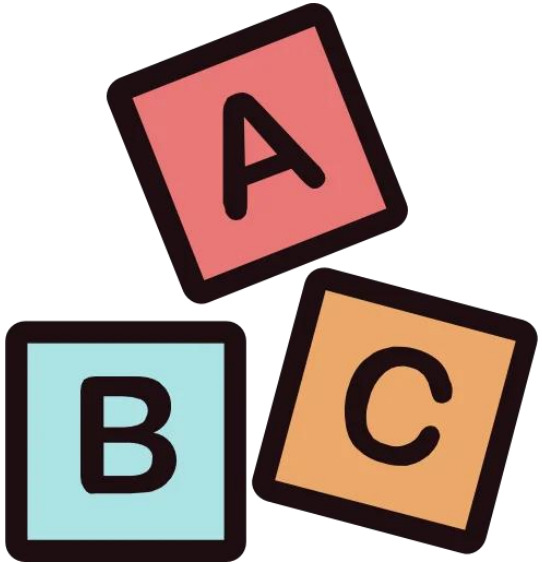
```
>>> '{x}/{y}/{z}'.format(x='foo', y='bar', z='baz')
'foo/bar/baz'
```

Use of F-strings in Python

Fstrings in Python

PEP 498 proposed to add a new string formatting mechanism: Literal String Interpolation. In this PEP, such strings will be referred to as “f-strings”, taken from the leading character used to denote such strings, and standing for “formatted strings”. Lets see about Use of F-strings in Python.

F-strings provide a way to embed expressions inside string literals, using a minimal syntax. It should be noted that an f-string is really an expression evaluated at run time, not a constant value. In Python source code, an f-string is a literal string, prefixed with ‘f’, which contains expressions inside braces.



String Format

- Python supports multiple ways to format text strings. These include %-formatting , str.format() , and string.template. All of them have their on advantages but in addition have it’s own disadvantages which creates problem during practice like we can not concat string and integers implicitly.
- The main Use of F-strings in Python is F-strings are faster than the two most commonly used string formatting mechanisms, which are %-formatting and str.format().
- Fstrings in Python provide a concise, readable way to include the value of Python expressions inside strings.

Example:

- w=70 , print(‘My weight is ‘ + w + ‘kgs’)
- TypeError: Can’t convert ‘int’ object to str implicitly

- w=70 #using f-strings
- print(f”My weight is {w} kgs.”)
- My weight is 70 kgs.

F-strings in Python

F-strings provide a concise, readable way to include the value of Python expressions inside strings.

```
Import datetime
date=datetime.datetime.Today()
Name='Prepsters'
print(f" Hello {Name} today is {date:%B %d}.")
```

Output: Hello Prepsters today is December 18 .

Code #1:

```
#f-strings
w=70
print(f“My weight is {w} kgs.”)
webs=‘PrepInsta’
print(f“{webs} is for Prepsters.”)
```

Output:
My weight is 70 kgs.
PrepInsta is for Prepsters.

Code #2:

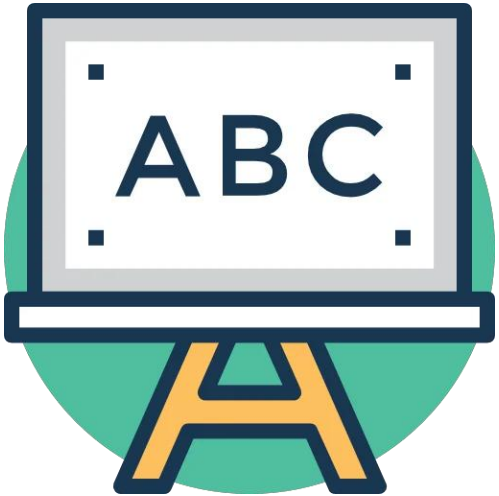
```
PI=22/7
print(f“Value of pi is {PI:.50F}“)
#OR
print(f“Value of pi is {22/7:.50f}“)
```

Output:
Value of pi is
3.14285714285714279370154144999105483293533325195312

Value of pi is
3.14285714a285714279370154144999105483293533325195312

String Interpolation in Python

String Interpolation
String interpolation is a process of injecting value into a placeholder (a placeholder is nothing but a variable to which you can assign data/value later) in a string literal. It helps in dynamically formatting the output in a fancier way. Python supports multiple ways to format string literals.



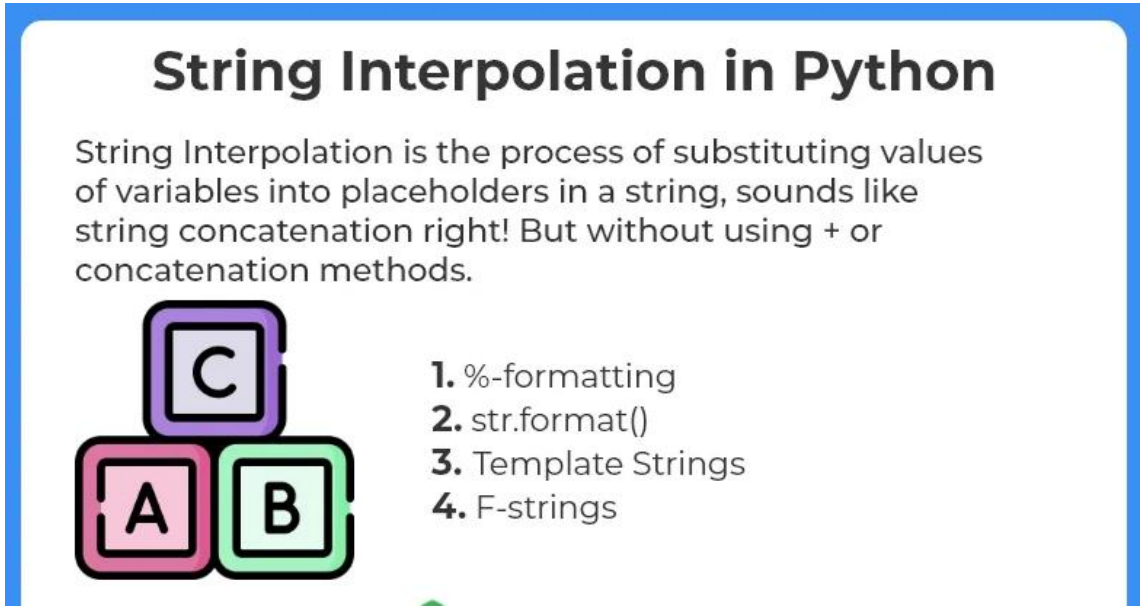
String Interpolation in Python

Ways of string interpolation

- %-formatting
- str.format()
- Template Strings
- F-strings

%-formatting

- In python modulo(%) operator is overloaded in str class to perform string formatting.
- If we have lots of variable to concat , then we can use %-formatting.



Decision making is required when we want to execute a code only if a certain condition is satisfied.

The `if…elif…else` statement is used in Python for decision making.

Python if Statement Syntax

```
if test expression:
    statement(s)
```

Here, the program evaluates the `test expression` and will execute statement(s) only if the test expression is `True`.

If the test expression is `False`, the statement(s) is not executed.

In Python, the body of the `if` statement is indicated by the indentation. The body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as `True`. `None` and `0` are interpreted as `False`.

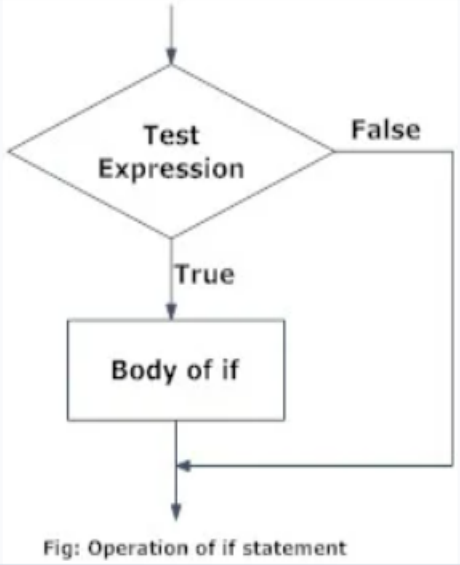


Fig: Operation of if statement

Flowchart of if statement in Python programming

Example: Python if Statement

```
# If the number is positive, we print an appropriate message

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")

num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

When you run the program, the output will be:

```
3 is a positive number
This is always printed
This is also always printed.
```

In the above example, `num > 0` is the test expression.

The body of `if` is executed only if this evaluates to `True`.

When the variable `num` is equal to 3, test expression is true and statements inside the body of `if` are executed.

If the variable `num` is equal to -1, test expression is false and statements inside the body of `if` are skipped.

The `print()` statement falls outside of the `if` block (unindented). Hence, it is executed regardless of the test expression.

Python if...else Statement

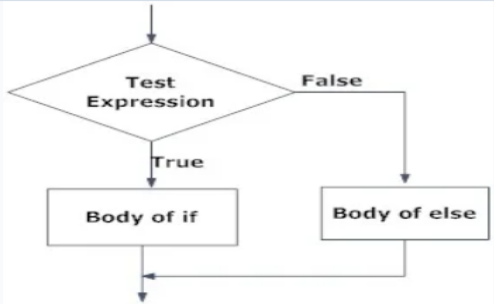
Syntax of if...else

```
if test expression:
    Body of if
else:
    Body of else
```

The `if..else` statement evaluates `test expression` and will execute the body of `if` only when the test condition is `True`.

If the condition is `False`, the body of `else` is executed. Indentation is used to separate the blocks.

Python if..else Flowchart



Example of if...else

```
# Program checks if the number is positive or negative
# And displays an appropriate message

num = 3

# Try these two variations as well.
# num = -5
# num = 0

if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

Output

```
Positive or Zero
```

Python if...elif...else Statement

Syntax of if...elif...else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

The `elif` is short for else if. It allows us to check for multiple expressions.

If the condition for `if` is `False`, it checks the condition of the next `elif` block and so on.

If all the conditions are `False`, the body of else is executed.

Only one block among the several `if...elif...else` blocks is executed according to the condition.

The `if` block can have only one `else` block. But it can have multiple `elif` blocks.

Flowchart of if...elif...else

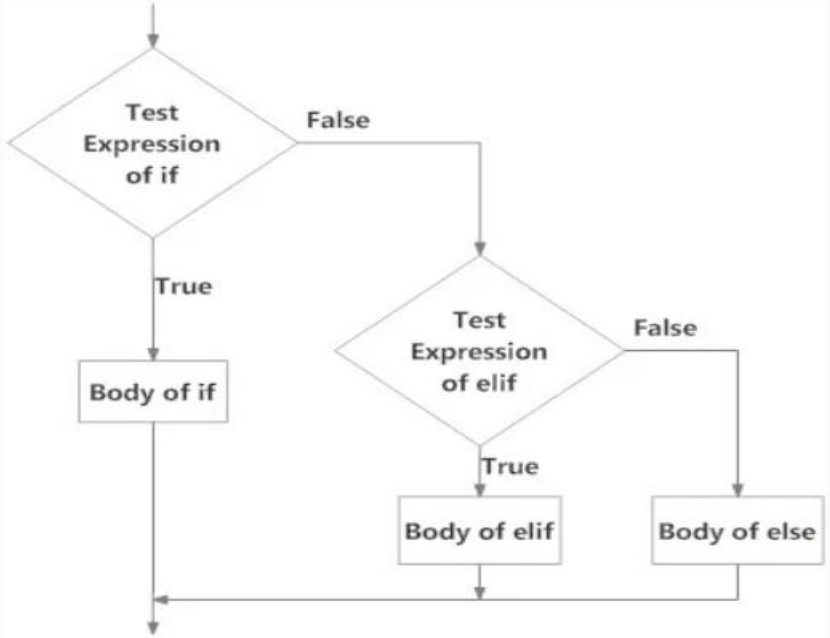


Fig: Operation of if...elif...else statement

Flowchart of if...elif....else statement in Python

Example of if...elif...else

```
'''In this program,
we check if the number is positive or
negative or zero and
display an appropriate message'''

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Python Nested if statements

We can have a `if...elif...else` statement inside another `if...elif...else` statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. They can get confusing, so they must be avoided unless necessary.

Python Nested if Example

```
'''In this program, we input a number
check if the number is positive or
negative or zero and display
an appropriate message
This time we use nested if statement'''

num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 5
Positive number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0
Zero
```

What is for loop in Python?

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
    loop body
```

Here, `val` is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Flowchart of for Loop

Fig: operation of for loop

Flowchart of for Loop in Python

Example: Python for Loop

```
# Program to find the sum of all numbers stored in a list

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

When you run the program, the output will be:

```
The sum is 48
```

The range() function

We can generate a sequence of numbers using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as `range(start, stop,step_size)`. step_size defaults to 1 if not provided.

The `range` object is "lazy" in a sense because it doesn't generate every number that it "contains" when we create it. However, it is not an iterator since it supports `in`, `len` and `__getitem__` operations.

This function does not store all the values in memory; it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function `list()`.

The following example will clarify this.

```
print(range(10))
print(list(range(10)))
print(list(range(2, 8)))
print(list(range(2, 20, 3)))
```

Output

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7]
[2, 5, 8, 11, 14, 17]
```

We can use the `range()` function in `for` loops to iterate through a sequence of numbers. It can be combined with the `len()` function to iterate through a sequence using indexing. Here is an example.

```
# Program to iterate through a list using indexing

genre = ['pop', 'rock', 'jazz']

# iterate over the list using index
for i in range(len(genre)):
    print("I like", genre[i])
```

Output

```
I like pop
I like rock
I like jazz
```

Python while Loop

What is while loop in Python?

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know the number of times to iterate beforehand.

Syntax of while Loop in Python

```
while test_expression:
    Body of while
```

In the while loop, test expression is checked first. The body of the loop is entered only if the `test_expression` evaluates to `True`. After one iteration, the test expression is checked again. This process continues until the `test_expression` evaluates to `False`.

In Python, the body of the while loop is determined through indentation.

The body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as `True`. `None` and `0` are interpreted as `False`.

Flowchart of while Loop

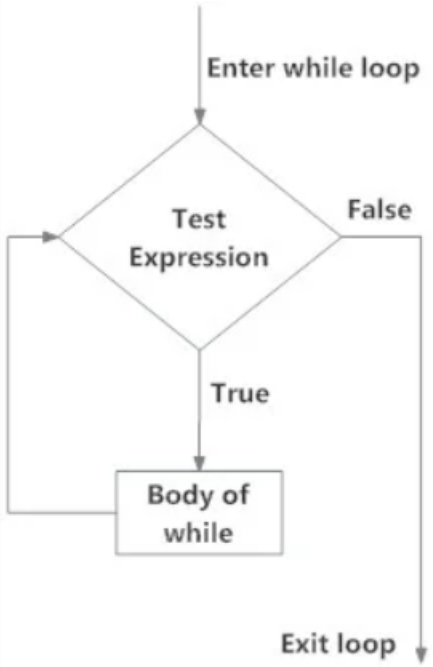


Fig: operation of while loop

Flowchart for while loop in Python

Example: Python while Loop

```
# Program to add natural
# numbers up to
# sum = 1+2+3+...+n

# To take input from the user,
# n = int(input("Enter n: "))

n = 10

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

When you run the program, the output will be:

```
Enter n: 10
The sum is 55
```

While loop with else

The `else` part is executed if the condition in the while loop evaluates to `False`.

The while loop can be terminated with a break statement. In such cases, the `else` part is ignored. Hence, a while loop's `else` part runs if no break occurs and the condition is false.

Here is an example to illustrate this.

```
'''Example to illustrate
the use of else statement
with the while loop'''

counter = 0

while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
```

Output

```
Inside loop
Inside loop
Inside loop
Inside else
```

Python break and continue

What is the use of break and continue in Python?

In Python, `break` and `continue` statements can alter the flow of a normal loop.

Loops iterate over a block of code until the test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.

The `break` and `continue` statements are used in these cases.

Python break statement

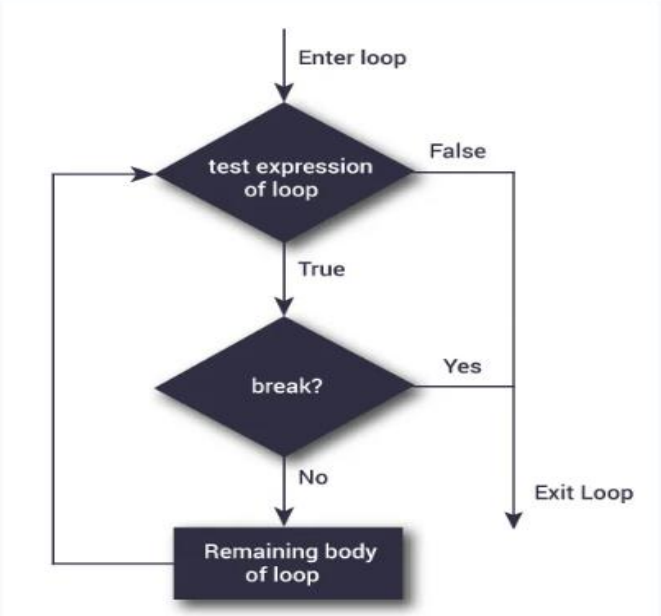
The `break` statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If the `break` statement is inside a nested loop (loop inside another loop), the `break` statement will terminate the innermost loop.

Syntax of break

```
break
```

Flowchart of break



Flowchart of break statement in Python

Example: Python break

```
# Use of break statement inside the loop

for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

Output

```
s
t
r
The end
```

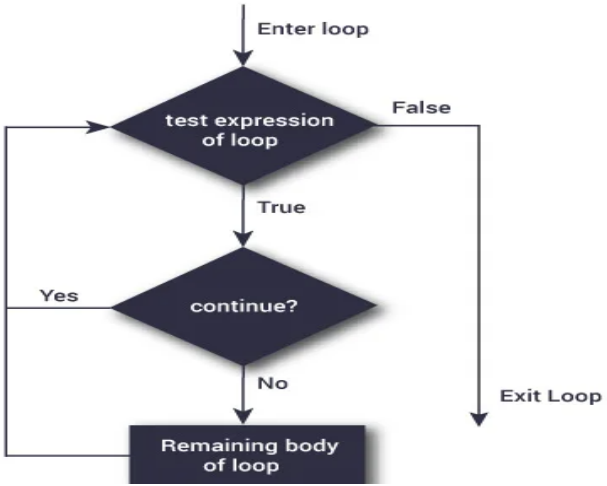
Python continue statement

The `continue` statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Syntax of Continue

```
continue
```

Flowchart of continue



Example: Python continue

```
# Program to show the use of continue statement inside loops

for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

Output

```
s
t
r
n
g
The end
```

Python zip()

The `zip()` function takes iterables (can be zero or more), aggregates them in a tuple, and returns it.

Example

```
languages = ['Java', 'Python', 'JavaScript']
versions = [14, 3, 6]

result = zip(languages, versions)
print(list(result))

# Output: [('Java', 14), ('Python', 3), ('JavaScript', 6)]
```

Syntax of zip()

The syntax of the `zip()` function is:

```
zip(*iterables)
```

zip() Parameters

Parameter	Description
`iterables`	**can be built-in iterables (like: list, string, dict), or user-defined iterables**

zip() Return Value

The `zip()` function returns an iterator of tuples based on the iterable objects.

- If we do not pass any parameter, `zip()` returns an empty iterator
- If a single iterable is passed, `zip()` returns an iterator of tuples with each tuple having only one element.
- If multiple iterables are passed, `zip()` returns an iterator of tuples with each tuple having elements from all the iterables.

  Suppose, two iterables are passed to `zip()`; one iterable containing three and other containing five elements. Then, the returned iterator will contain three tuples. It's because the iterator stops when the shortest iterable is exhausted.

Example 1: Python zip()

```
number_list = [1, 2, 3]
str_list = ['one', 'two', 'three']

# No iterables are passed
result = zip()

# Converting iterator to list
result_list = list(result)
print(result_list)

# Two iterables are passed
result = zip(number_list, str_list)

# Converting iterator to set
result_set = set(result)
print(result_set)
```

Output

```
[]
{(2, 'two'), (3, 'three'), (1, 'one')}
```

Example 2: Different number of iterable elements

```
numbersList = [1, 2, 3]
str_list = ['one', 'two']
numbers_tuple = ('ONE', 'TWO', 'THREE', 'FOUR')

# Notice, the size of numbersList and numbers_tuple is different
result = zip(numbersList, numbers_tuple)

# Converting to set
result_set = set(result)
print(result_set)

result = zip(numbersList, str_list, numbers_tuple)

# Converting to set
result_set = set(result)
print(result_set)
```

Output

```
{(2, 'TWO'), (3, 'THREE'), (1, 'ONE')}
{(2, 'two', 'TWO'), (1, 'one', 'ONE')}
```

Example 3: Unzipping the Value Using zip()

```
coordinate = ['x', 'y', 'z']
value = [3, 4, 5]

result = zip(coordinate, value)
result_list = list(result)
print(result_list)

c, v =  zip(*result_list)
print('c =', c)
print('v =', v)
```

Output

```
[('x', 3), ('y', 4), ('z', 5)]
c = ('x', 'y', 'z')
v = (3, 4, 5)
```

Python enumerate()

The `enumerate()` method adds a counter to an iterable and returns it (the enumerate object).

Example

```
languages = ['Python', 'Java', 'JavaScript']

enumerate_prime = enumerate(languages)

# convert enumerate object to list
print(list(enumerate_prime))

# Output: [(0, 'Python'), (1, 'Java'), (2, 'JavaScript')]
```

Syntax of enumerate()

The syntax of `enumerate()` is:

```
enumerate(iterable, start=0)
```

enumerate() Parameters

`enumerate()` method takes two parameters:

- **iterable** - a sequence, an iterator, or objects that supports iteration
- **start** (optional) - `enumerate()` starts counting from this number. If `start` is omitted, `0` is taken as `start`.

---

enumerate() Return Value

`enumerate()` method adds counter to an iterable and returns it. The returned object is an enumerate object.

You can convert enumerate objects to list and tuple using list() and tuple() method respectively.

Example 1: How enumerate() works in Python?

```
grocery = ['bread', 'milk', 'butter']
enumerateGrocery = enumerate(grocery)

print(type(enumerateGrocery))

# converting to list
print(list(enumerateGrocery))

# changing the default counter
enumerateGrocery = enumerate(grocery, 10)
print(list(enumerateGrocery))
```

Output

```
<class 'enumerate'>
[(0, 'bread'), (1, 'milk'), (2, 'butter')]
[(10, 'bread'), (11, 'milk'), (12, 'butter')]
```

Example 2: Looping Over an Enumerate object

```
grocery = ['bread', 'milk', 'butter']

for item in enumerate(grocery):
  print(item)

print('\n')

for count, item in enumerate(grocery):
  print(count, item)

print('\n')
# changing default start value
for count, item in enumerate(grocery, 100):
  print(count, item)
```

Output

```
(0, 'bread')
(1, 'milk')
(2, 'butter')

0 bread
1 milk
2 butter

100 bread
101 milk
102 butter
```