Python Concepts for InfyTQ Qualifying Round for 2022 Batch

by

TalentBattle

One Stop Platform for Placement Preparation

www.talentbattle.in

For Off-campus updates & Placement Preparation Join our Whatsapp group & Telegram channel here: https://bit.ly/3jyTDkg

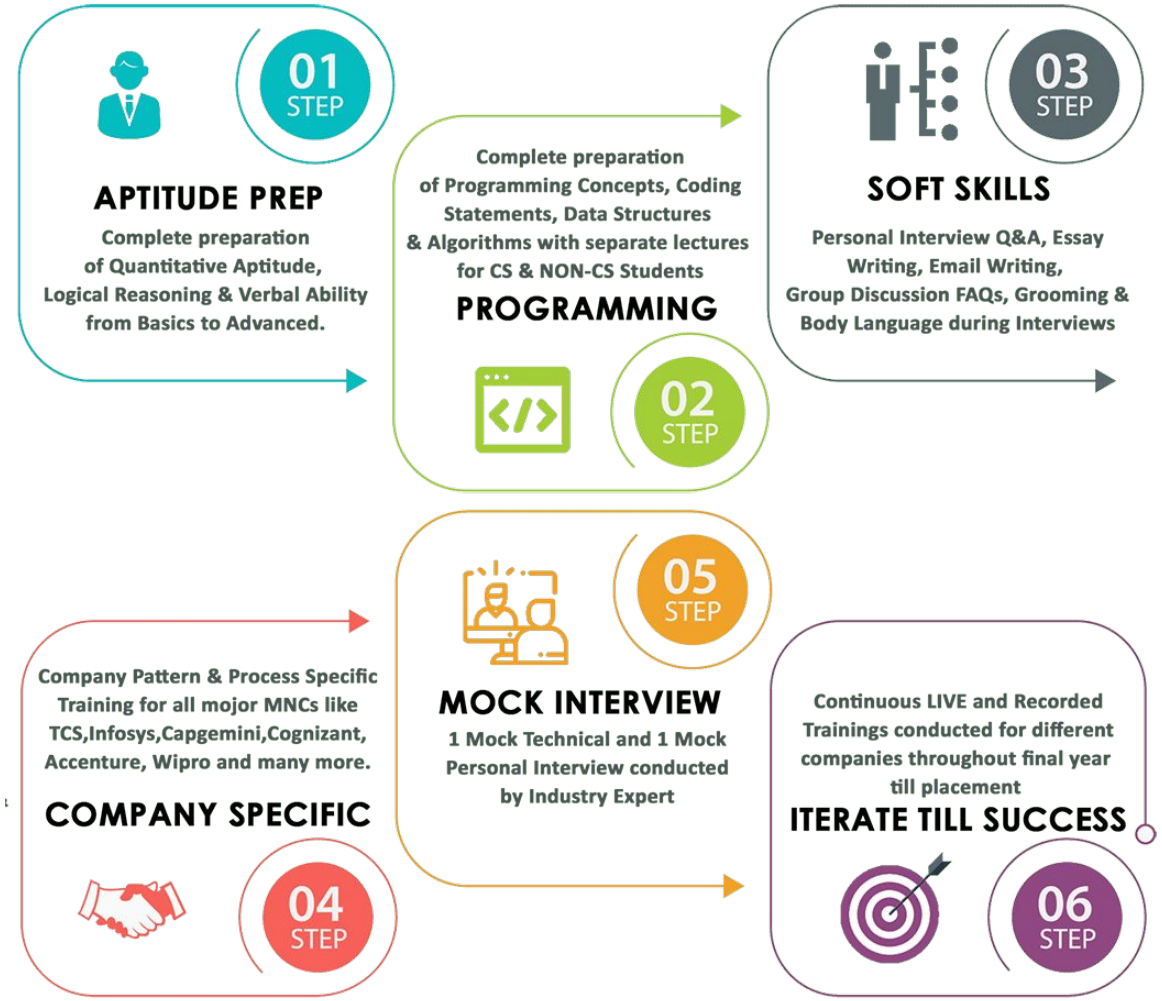
InfyTQ Concepts PDF Notes

Complete Placement
Preparatory LIVE
MasterClass

Complete Placement Preparatory LIVE Mastercla

Preparing for the upcoming placement season? Battle your way through the crowd with the best placement preparation course online.

- This course is a complete end-to-end solution to your hurdles in placement preparation with a perfect blend of **LIVE** & Recorded Classes.
- Aptitude(Quantitative Aptitude, Logical Reasoning, Verbal Ability), Programming concepts, Coding Statement Preparation, Personal and Technical Interview preparation is completely covered in this course
- In this course, our experts are well known for their teaching methods which will help to understand Aptitude, Coding, Programming concepts in the easiest and quickest possible way from basic to advanced level.
- Company Specific Training Modules required for TCS, Wipro, Infosys, Accenture, Cognizant, Capgemini, Mindtree, Deloitte, Tech Mahindra, LTI, IBM, Atos-Syntel, Persistent, Hexaware, and many more companies are covered in this course according to the latest pattern.



More Details and Registration Link: https://bit.ly/39sgQSM

INDEX

Exam Pattern: 20 Questions

Topics:

Decision Making, Loops, Linked List, Queue, Deque, Stack, Exception, Hash Function, Class Abstraction, Inheritance.

About Python Programming

Python is a powerful general-purpose programming language. It is used in web development, data science, creating software prototypes, and so on. Fortunately for beginners, Python has simple easy-to-use syntax. This makes Python an excellent language to learn to program for beginners.

•**Free and open-source** - You can freely use and distribute Python, even for commercial use.

•**Easy to learn** - Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like C++, Java, C#.

•**Portable** - You can move Python programs from one platform to another, and run it without any changes.

Python if...else Statement

What is if...else statement in Python?

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.

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.")
```

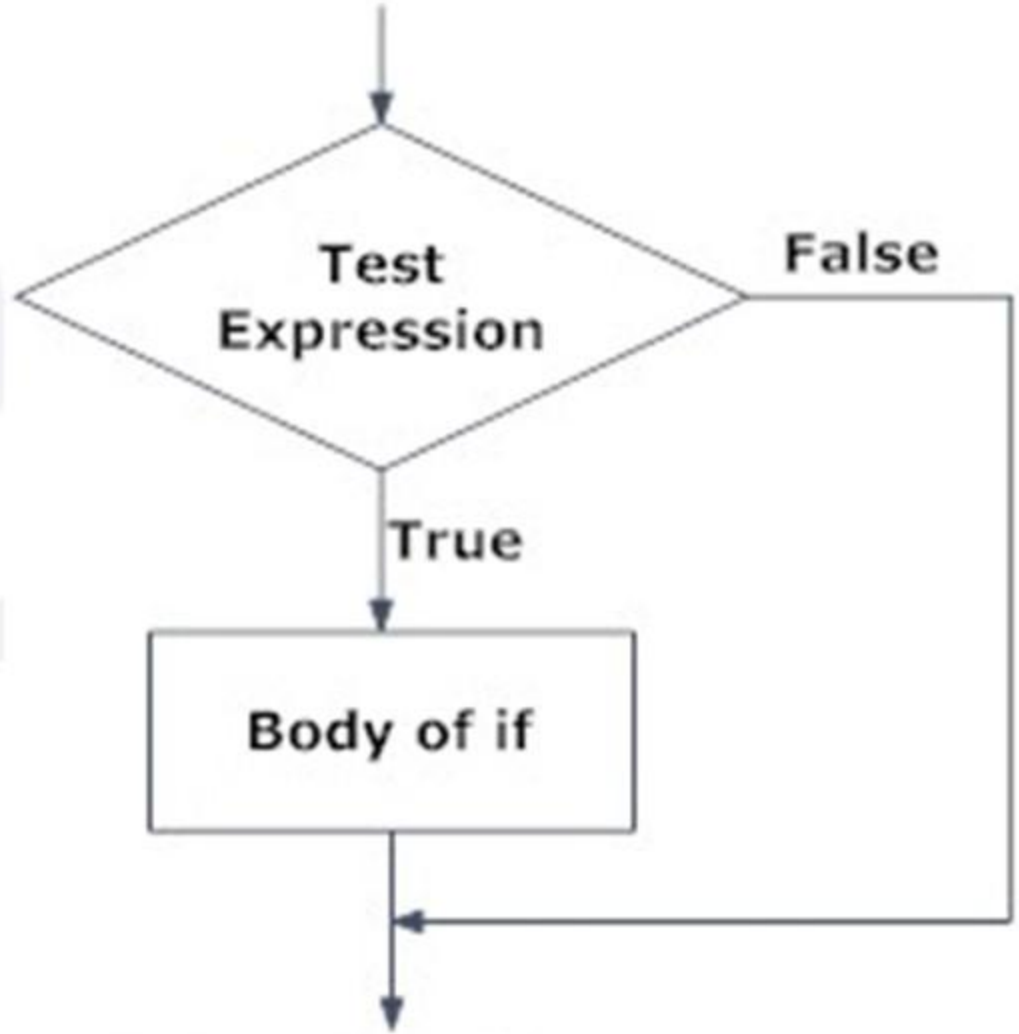


Fig: Operation of if statement

Python if..else Flowchart

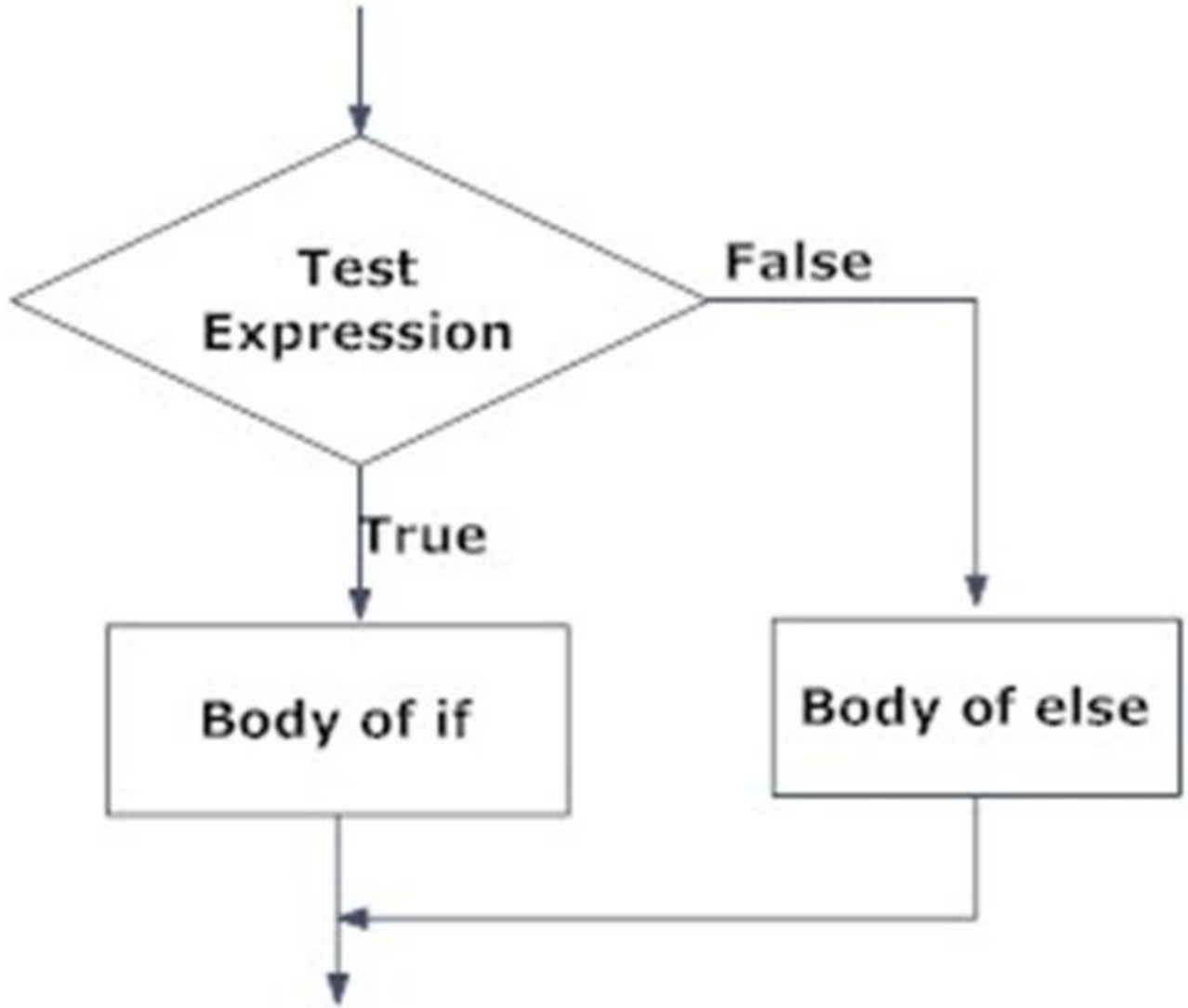


Fig: Operation of if...else statement

Python if...elif...else Statement

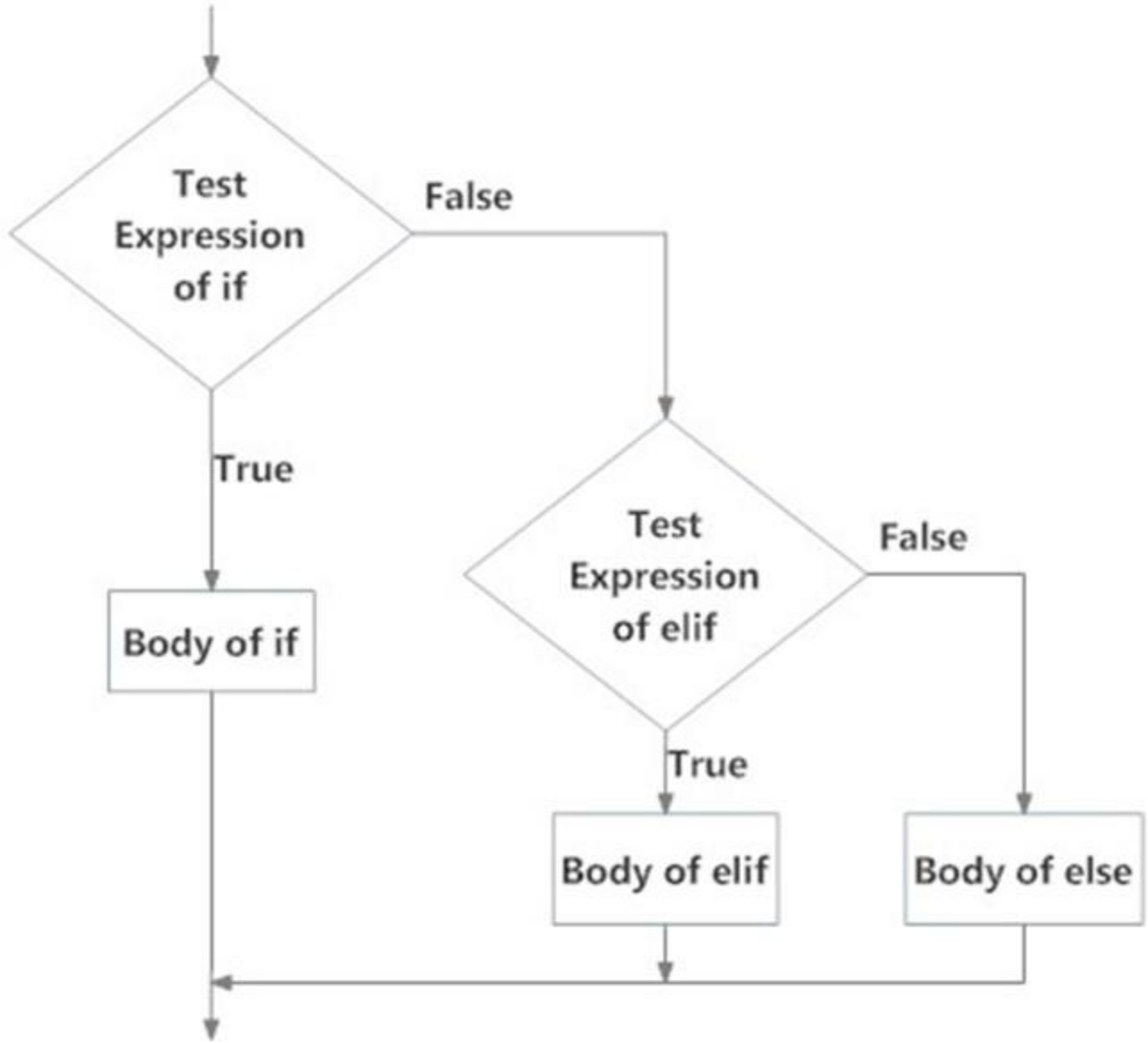


Fig: Operation of if...elif...else statement

Python for Loop

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:
        Body of for
```

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.

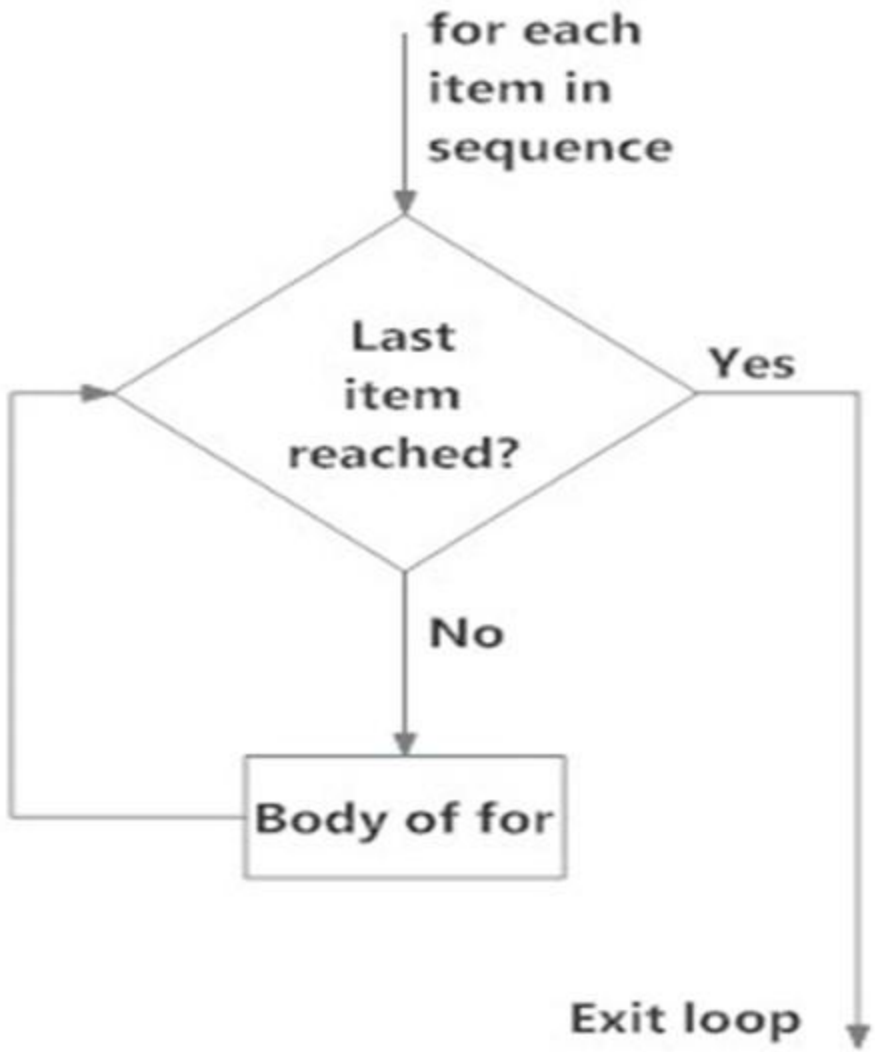


Fig: operation of for loop

for loop with else

A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.

The break keyword can be used to stop a for loop. In such cases, the else part is ignored.

Here is an example to illustrate this.

```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

When you run the program, the output will be:

```
0
1
5
No items left.
```

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.

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.

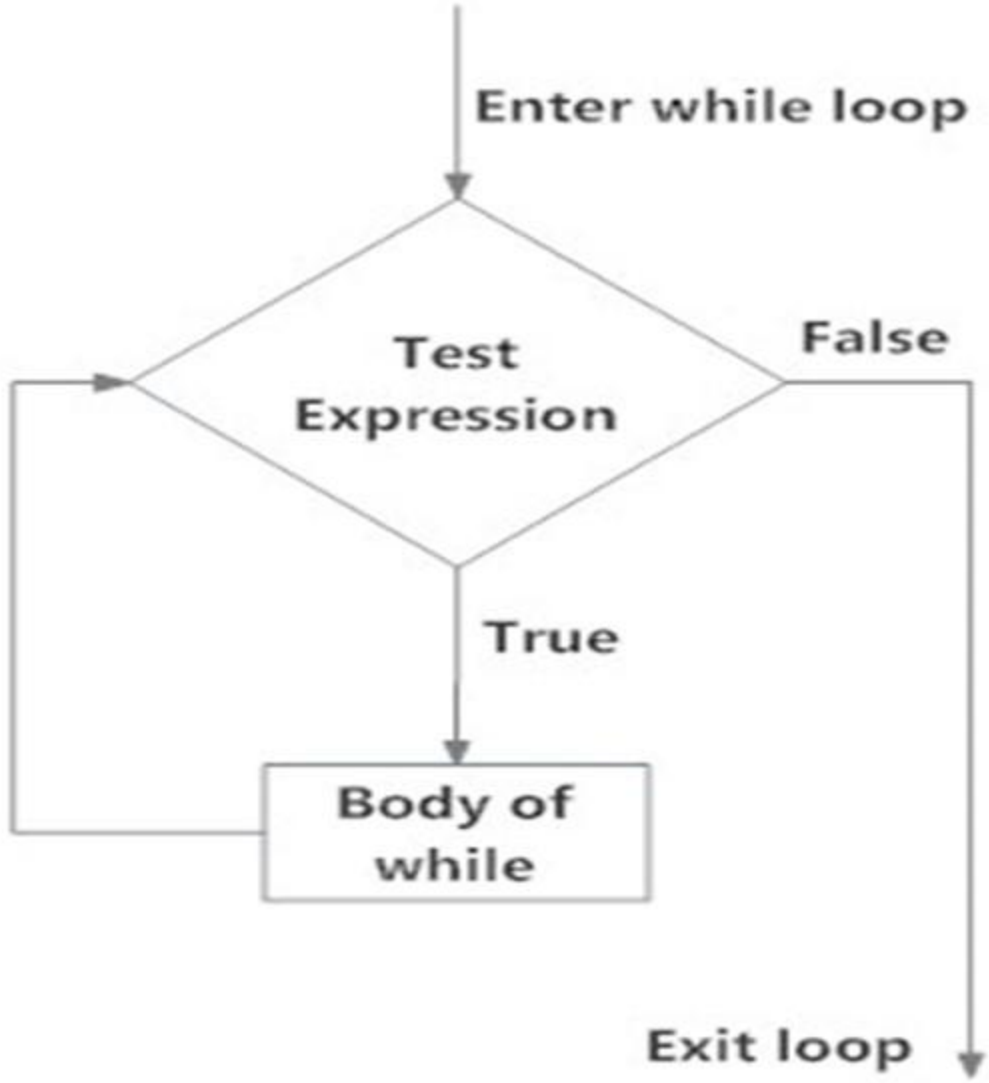


Fig: operation of while loop

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

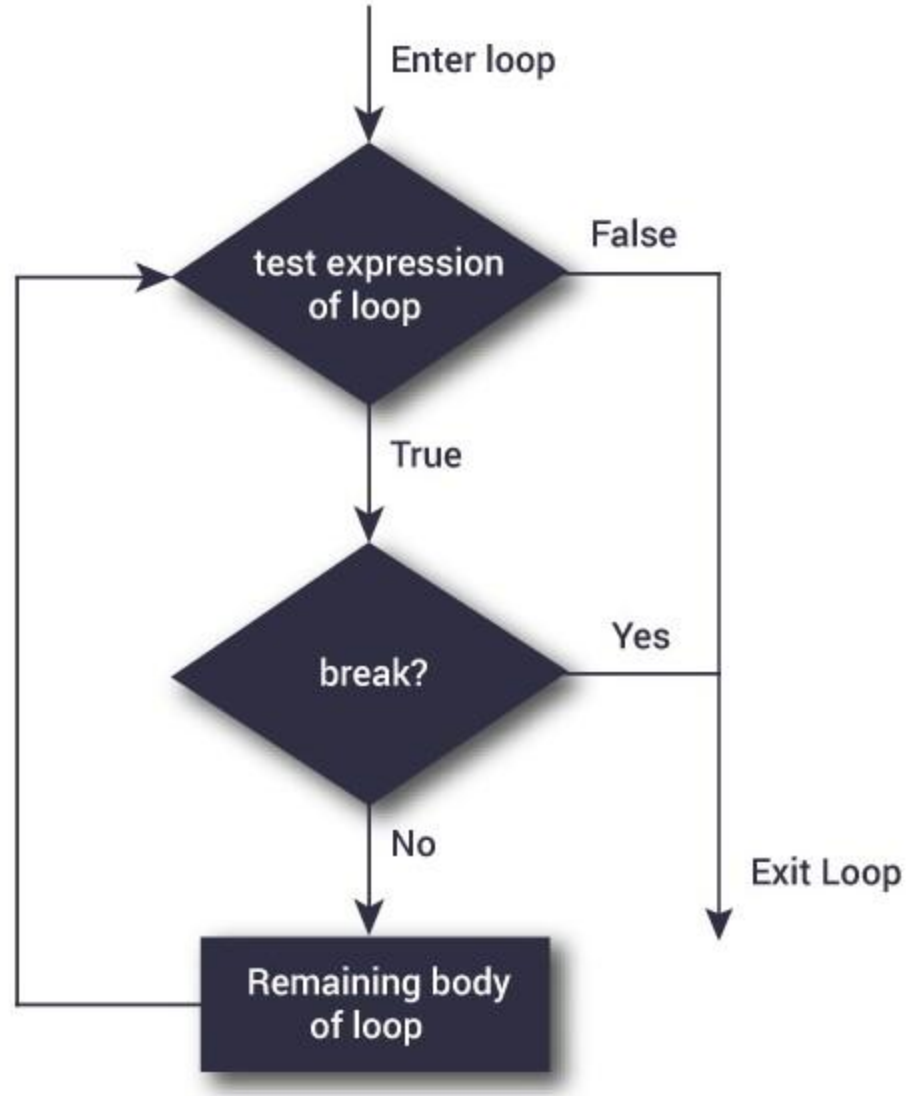
The break statement terminates the loop containing it.

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.

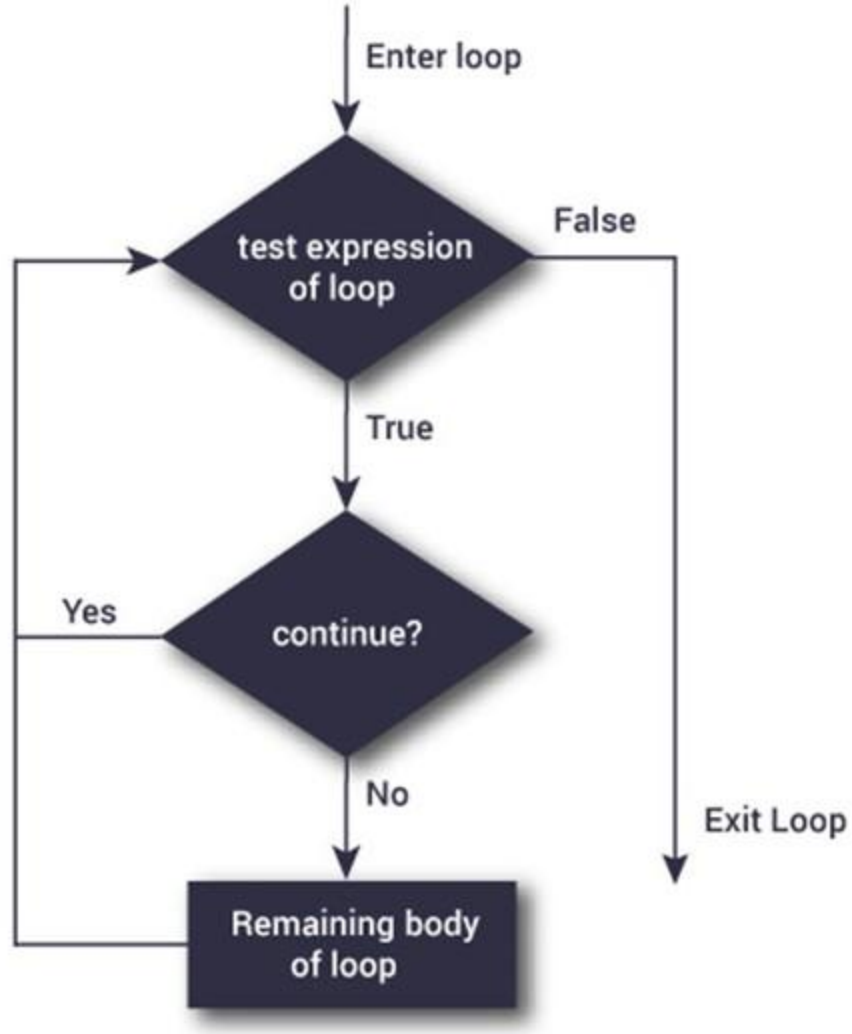
Syntax of break

break



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

Python Object Oriented Programming

Python is a multi-paradigm programming language. It supports different programming approaches.

One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

•attributes

•behavior

Let's take an example:

A parrot is can be an object, as it has the following properties:

•name, age, color as attributes

•singing, dancing as behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

Class

A class is a blueprint for the object.

We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, a parrot is an object.

The example for class of parrot can be :

```
class Parrot:
    pass
```

Here, we use the class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class.

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

obj = Parrot()

Here, obj is an object of class Parrot.

Suppose we have details of parrots. Now, we are going to show how to build the class and objects of parrots.

Key Points to Remember:

- Object-Oriented Programming makes the program easy to understand as well as efficient.
- Since the class is sharable, the code can be reused.
- Data is safe and secure with data abstraction.
- Polymorphism allows the same interface for different objects, so programmers can write efficient code.

Python Inheritance

Inheritance is a powerful feature in object oriented programming.

It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

Python Inheritance Syntax

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Derived class inherits features from the base class where new features can be added to it. This results in re-usability of code.

Python Multiple Inheritance

A class can be derived from more than one base class in Python, similar to C++. This is called multiple inheritance.

In multiple inheritance, the features of all the base classes are inherited into the derived class. The syntax for multiple inheritance is similar to single inheritance.

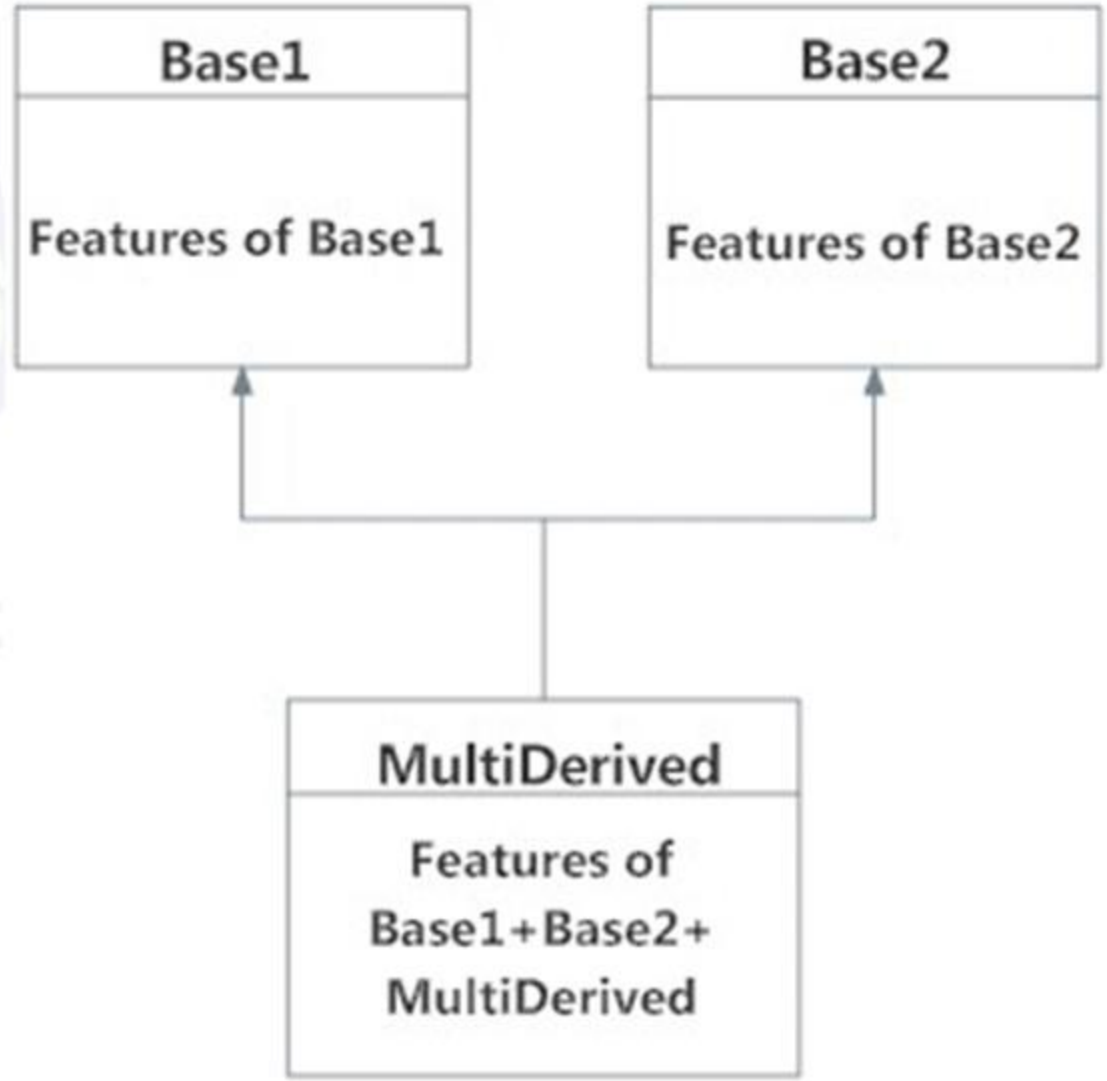
Example

```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass
```

Here, the MultiDerived class is derived from Base1 and Base2 classes.



Python Multilevel Inheritance

We can also inherit from a derived class. This is called multilevel inheritance. It can be of any depth in Python.

In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.

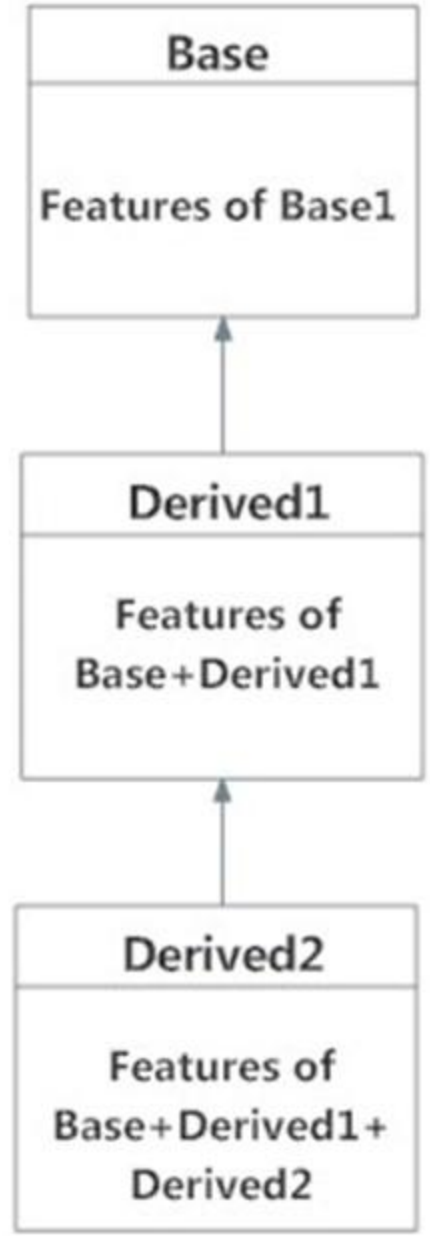
An example with corresponding visualization is given below.

```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

Here, the Derived1 class is derived from the Base class, and the Derived2 class is derived from the Derived1 class.



Method Resolution Order in Python

Every class in Python is derived from the object class. It is the most base type in Python.

So technically, all other classes, either built-in or user-defined, are derived classes and all objects are instances of the object class.

In the multiple inheritance scenario, any specified attribute is searched first in the current class. If not found, the search continues into parent classes in depth-first, left-right fashion without searching the same class twice.

MRO must prevent local precedence ordering and also provide monotonicity. It ensures that a class always appears before its parents. In case of multiple parents, the order is the same as tuples of base classes.

MRO of a class can be viewed as the __mro__ attribute or the mro() method. The former returns a tuple while the latter returns a list.

Python Errors and Built-in Exceptions

We can make certain mistakes while writing a program that lead to errors when we try to run it. A python program terminates as soon as it encounters an unhandled error. These errors can be broadly classified into two classes:

- **Syntax errors**
- **Logical errors (Exceptions)**

Python Logical Errors (Exceptions)

Errors that occur at runtime (after passing the syntax test) are called exceptions or logical errors.

For instance, they occur when we try to open a file(for reading) that does not exist (FileNotFoundError), try to divide a number by zero (ZeroDivisionError), or try to import a module that does not exist (ImportError).

Whenever these types of runtime errors occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

Python Built-in Exceptions

Illegal operations can raise exceptions. There are plenty of built-in exceptions in Python that are raised when corresponding errors occur. We can view all the built-in exceptions using the built-in local() function as follows:

```
print(dir(locals()['__builtins__']))
```

locals()['__builtins__'] will return a module of built-in exceptions, functions, and attributes. dir allows us to list these attributes as strings.

Some of the common built-in exceptions in Python programming along with the error that cause them are listed below:

Exception	Cause of Error
AssertionError	Raised when an assert statement fails.
AttributeError	Raised when attribute assignment or reference fails.
EOFError	Raised when the input() function hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raise when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when the index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+C or Delete).
MemoryError	Raised when an operation runs out of memory.

NameError	Raised when a variable is not found in local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when system operation causes system related error.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by next() function to indicate that there is no further item to be returned by iterator.
SyntaxError	Raised by parser when syntax error is encountered.
IndentationError	Raised when there is incorrect indentation.
TabError	Raised when indentation consists of inconsistent tabs and spaces.

SystemError	Raised when interpreter detects internal error.
SystemExit	Raised by sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translating.
ValueError	Raised when a function gets an argument of correct type but improper value.
ZeroDivisionError	Raised when the second operand of division or modulo operation is zero.

Python Exception Handling Using try, except and finally statement

Exceptions in Python

Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).

When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

For example, let us consider a program where we have a function A that calls function B, which in turn calls function C. If an exception occurs in function C but is not handled in C, the exception passes to B and then to A.

If never handled, an error message is displayed and our program comes to a sudden unexpected halt.

Catching Exceptions in Python

In Python, exceptions can be handled using a try statement.

The critical operation which can raise an exception is placed inside the try clause. The code that handles the exceptions is written in the except clause.

We can thus choose what operations to perform once we have caught the exception.

Python try with else clause

In some situations, you might want to run a certain block of code if the code block inside try ran without any errors. For these cases, you can use the optional else keyword with the try statement.

Note: Exceptions in the else clause are not handled by the preceding except clauses.

Python try...finally

The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.

For example, we may be connected to a remote data center through the network or working with a file or a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource before the program comes to a halt whether it successfully ran or not. These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee the execution.

Abstract Classes in Python

An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has a declaration but does not have an implementation. While we are designing large functional units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class.

Why use Abstract Base Classes :

By defining an abstract base class, you can define a common Application Program Interface(API) for a set of subclasses. This capability is especially useful in situations where a third-party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code-base where keeping all classes in your mind is difficult or not possible.

How Abstract Base classes work :

By default, Python does not provide abstract classes. Python comes with a module that provides the base for defining Abstract Base classes(ABC) and that module name is ABC. ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword @abstractmethod.

Python - Linked Lists

A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter. We have already seen how we create a node class and how to traverse the elements of a node. In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

Creation of Linked list

A linked list is created by using the node class. We create a Node object and create another class to use this ode object. We pass the appropriate values through the node object to point the to the next data elements.

Traversing a Linked List

Singly linked lists can be traversed in only forward direction starting form the first data element. We simply print the value of the next data element by assigning the pointer of the next node to the current data element.

Insertion in a Linked List

Inserting element in the linked list involves reassigning the pointers from the existing nodes to the newly inserted node. Depending on whether the new data element is getting inserted at the beginning or at the middle or at the end of the linked list, we have the below scenarios.

Inserting at the Beginning of the Linked List

This involves pointing the next pointer of the new data node to the current head of the linked list. So the current head of the linked list becomes the second data element and the new node becomes the head of the linked list.

Inserting at the End of the Linked List

This involves pointing the next pointer of the current last node of the linked list to the new data node. So the current last node of the linked list becomes the second last data node and the new node becomes the last node of the linked list.

Inserting in between two Data Nodes

This involves changing the pointer of a specific node to point to the new node. That is possible by passing in both the new node and the existing node after which the new node will be inserted. So we define an additional class which will change the next pointer of the new node to the next pointer of middle node. Then assign the new node to next pointer of the middle node.

Python - Stack

PUSH from a Stack

In a stack the element inserted last in sequence will come out first as we can remove only from the top of the stack. Such feature is known as Last in First Out(LIFO) feature. The operations of adding and removing the elements is known as PUSH and POP. We declare an empty list and use the append() and pop() methods to add and remove the data elements.

POP from a Stack

As we know we can remove only the top most data element from the stack, we implement a python program which does that. The remove function returns the top most element. we check the top element by calculating the size of the stack first and then use the in-built pop() method to find out the top most element.

Python - Queue

We are familiar with queue in our day to day life as we wait for a service. The queue data structure also means the same where the data elements are arranged in a queue. The uniqueness of queue lies in the way items are added and removed. The items are allowed at on end but removed form the other end. So it is a First-in-First out method. An queue can be implemented using python list where we can use the insert() and pop() methods to add and remove elements. Their is no insertion as data elements are always added at the end of the queue.

- Adding Elements to a Queue
- Removing Element from a Queue

Python - Dequeue

A double-ended queue, or deque, supports adding and removing elements from either end. The more commonly used stacks and queues are degenerate forms of deques, where the inputs and outputs are restricted to a single end.

Python - Hash Table

Hash tables are a type of data structure in which the address or the index value of the data element is generated from a hash function. That makes accessing the data faster as the index value behaves as a key for the data value. In other words Hash table stores key-value pairs but the key is generated through a hashing function.

So the search and insertion function of a data element becomes much faster as the key values themselves become the index of the array which stores the data.

In Python, the Dictionary data types represent the implementation of hash tables. The Keys in the dictionary satisfy the following requirements.

•The keys of the dictionary are hash able i.e. the are generated by hashing function which generates unique result for each unique value supplied to the hash function.

•The order of data elements in a dictionary is not fixed.

Python Recursion

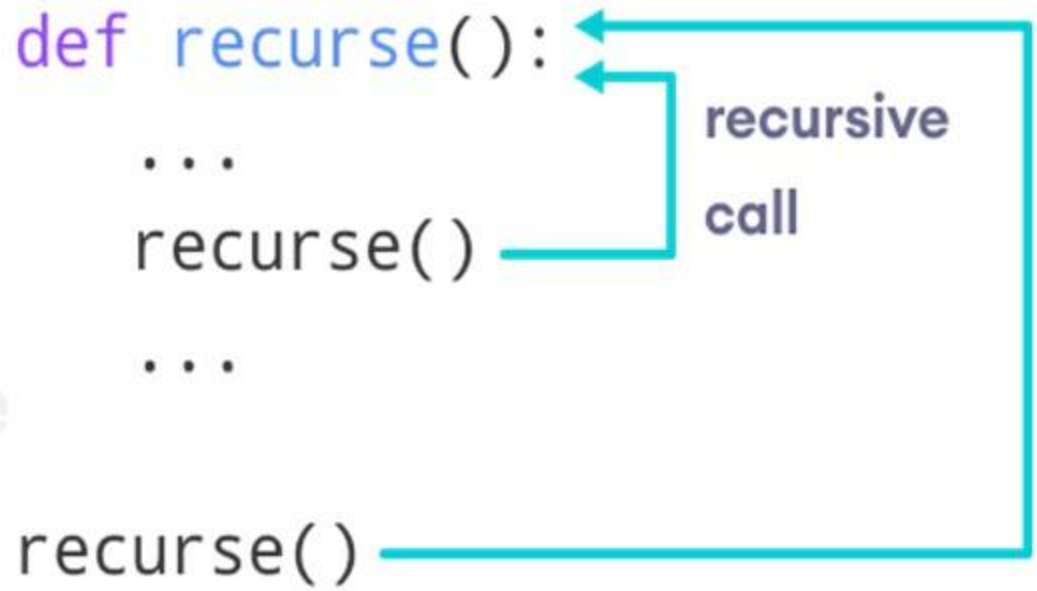
What is recursion?
Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

Python Recursive Function
In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.
Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

The Python interpreter limits the depths of recursion to help avoid infinite recursions, resulting in stack overflows.

By default, the maximum depth of recursion is 1000. If the limit is crossed, it results in RecursionError.



THANK YOU!!!