



#1

LIMITED  
EDITION

VOL 413

# PYTHON INTERVIEW QUESTIONS

"STAY UPDATED, STAY AHEAD"

TechVidya is ISO Certified 9001:2015 accredited EdTech Company registered with ROC under the companies act 1956, offers self-paced, online and offline programs. TechVidya pedagogy is rooted in the principle that every young mind should be equipped with exceptional knowledge and skills that benefit them in real life.

YEAR  
2023

---

Company Info

**TechVidya**  
EdTech Company

Company Website

**techvidya.education**  
Official Website

Company Contact

**+91 83759 66700**  
Official Number

# Preface

Python is one of the most influential and fastest evolving languages in the field of software development. It is also an important part of curriculum of computer science undergraduate students. Therefore, it is a great idea to pursue a career in Python Programming.

Interviews are very different from academics. In an interview, apart from the text book knowledge and practical understanding, the approach to solving the problem is very crucial. To prepare for Python interview, it is critical that your knowledge of the subject is effectively communicated to the interviewer.

This book has been written with the objective of helping readers to prepare for exam or an interview. It contains probable questions and their solutions. I have compiled this book on Python, the way I would myself prepare for an exam or an interview. Over preparation can be overwhelming especially for students who are preparing for exams or interviewing for their first job. This is your guide to success !!

The book is organized as follows:

It is divided into two sections:

## (1) Basic Python Programming

This section consists of seven chapters as follows:

- Chapter 1 • Introduction to Python
- Chapter 2 • Data Types and Their in-built functions
- Chapter 3 • Operators in Python
- Chapter 4 • Decision Making & Loops.
- Chapter 5 • User Defined Functions.
- Chapter 6 • Classes and Inheritance. Finally,
- Chapter 7 • Files

## (2) Data Structures and Algorithms

The next 7 chapters of the book come under this category:

- Chapter 8 Algorithm Analysis and Big O.

iv - *Python Interview Questions*

- Chapter 9 Array Sequences
- Chapter 10 Stacks, Queues and Deque.
- Chapter 11 Linked Lists.
- Chapter 12 Concept of Recursion.
- Chapter 13 Trees
- Chapter 14 Searching and Sorting

The book not only gives you a strong understanding of foundations but also teaches you how to take it one step further. It has been written in simple language and my objective is to explain the logic behind every concept. Students and professionals at large will benefit from this book.

In a Python Programming interview you would be tested for basics, logical reasoning and problem solving skills. How you look at the problem, analyse it and code is all part of the test.



# Foreword

Python is a vast subject and if you have to prepare for an interview in a short span of time you may feel a bit lost or overwhelmed about how to go about preparing for the day. From my personal experience, I know that there is huge difference in learning a programming language in a classroom and its implementation in real time projects. I have, therefore, worked on this book to create a training material that can work as a lifelong companion for both students as well as professionals.

Programming language interviews can be tricky and so having strong foundations is very important. A technical interview starts as a simple discussion and then you would be asked questions randomly from different topics. The best way to prepare for such an interview is to follow a systematic approach.

In this book, the content is organized in a very systematic way. The content is divided into two sections – Python programming basics and Python Data Structures and Algorithms. Even if you are good in programing I would suggest you not to take the first section lightly. A lot of emphasis is given to the basics because even a small mistake in programming basics is unacceptable at any stage. Again content is organized in a specific order providing salient points for each topic followed by various types of questions related to the topic. Solutions for all questions are also enumerated.

The second section, Data Structures and Algorithms deals with very important chapters. I have provided step by step logical explanation which will help you understand the topics. The shift from simple to complex topics is gradual so that it is easy for you to grasp the subject.

I cannot emphasise enough on the importance of hands-on coding. It helps you get a better understanding of the subject. I request the readers to work out all the exercises on your systems. You can effectively explain a logic in an interview only if you have worked out the problems yourself !

While preparing for a Python interview just focus on the subject. Nothing else matters. A good interviewer will never let a good programmer go.

# Table of Contents

## SECTION I : PYTHON BASICS

Chapter 1 Introduction to Python	1
Chapter 2 Data Types and Their in-built Functions	19
Chapter 3 Operators in Python	67
Chapter 4 Decision Making and Loops	79
Chapter 5 User Defined Functions	87
Chapter 6 Classes and Inheritance	101
Chapter 7 Files	111

## SECTION II: PYTHON DATA STRUCTURE

### AND ALGORITHM

Chapter 8	Algorithm Analysis and Big-O	117
Chapter 9	Array Sequence	131
Chapter 10	Stacks, Queues, and Deque	149
Chapter 11	Linked List	171
Chapter 12	Recursion	195
Chapter 13	Trees	205
Chapter 14	Searching and Sorting	241

# Chapter 1

## Introduction to Python

### Python

- Python is a very popular programming language. It is known for being an interactive and object oriented programming language.
- Free software, open source language with huge number of volunteers who are working hard to improve it. This is the main reason why the language is current with the newest trends.
- It has several libraries which help build powerful code in short span of time.
- It is a very simple, powerful, and general purpose computer programming language.
- Python is easy to learn and easy to implement.
- Well known corporations are using Python to build their site. Some of the well known websites built in Python are as follows:

® Google ® YouTube

® Quora ® Dropbox

® Yahoo! ® Reddit

® Instagram ® Spotify

® Survey Monkey ® Bitly

- Main reason for popularity of Python programming language is simplicity of the code.
- You require no skills to learn Python.

### Question: What can you do with python?

Answer: There is no limit to what can be achieved with the help of Python Programming:

- Python can be used for small or large, online or offline applications.
- Developers can code using fewer lines of code compared to other languages.

## 2 - Python Interview Questions

- Python is widely used for developing web applications as it has a dynamic system and automatic memory management is one of its strongest points.
- Some of the very well-known Python framework are: Pyramid, Django, and Flask.
- Python is also used for simple scripting and scientific modelling and big data applications:
- It is the number one choice for several data scientists.
- Its libraries such as NumPy, Pandas data visualization libraries such as Matplotlib and Seaborn have made Python very popular in this field.
- Python also has some interesting libraries such as Scikit-Learn, NLTK, and TensorFlow that implement Machine learning Algorithms.
- Video Game can be created using PyGame module. Such applications can run on Android devices.
- Python can be used for web scrapping.
- Selenium with Python can be used for things like opening a browser or posting a status on Facebook.
- Modules such as Tkinter and PyQt allow you to build a GUI desktop application.

**Question: Why is Python considered to be a highly versatile programming language?**

*Answer:* Python is considered to be a highly versatile programming language because it supports multiple models of programming such as:

- OOP
- Functional
- Imperative
- Procedural

**Question: What are the advantages of choosing Python over any other programming language?**

*Answer:* The advantages of selecting Python over other programming languages are as follows:

- Extensible in C and C++.
- It is dynamic in nature.

- Easy to learn and easy to implement.
- Third party operating modules are present: As the name suggests a third party module is written by third party which means neither you nor the python writers have developed it. However, you can make use of these modules to add functionality to your code.

**Question:** What do you mean when you say that Python is an interpreted language?

**Answer:** When we say that Python is an interpreted language it means that python code is not compiled before execution. Code written in compiled languages such as Java can be executed directly on the processor because it is compiled before runtime and at the time of execution it is available in the form of machine language that the computer can understand. This is not the case with Python. It does not provide code in machine language before runtime. The translation of code to machine language occurs while the program is being executed.

**Question:** Are there any other interpreted languages that you have heard of?

**Answer:** Some frequently used interpreted programming languages are as follows:

- Python
- Pearl
- JavaScript
- PostScript
- PHP
- PowerShell

**Question:** Is Python dynamically typed?

**Answer:** Yes, Python is dynamically typed because in a code we need not specify the type of variables while declaring them. The type of a variable is not known until the code is executed.

**Question:** Python is a high level programming language? What is a need for high level programming languages?

**Answer:** High level programming languages act as a bridge between the machine and humans. Coding directly in machine language can be a very time consuming and cumbersome process and it would definitely restrict coders from achieving their goals. High level programming languages

#### 4 - Python Interview Questions

like Python, JAVA, C++, etc are easy to understand. They are tools which the programmers can use for advanced level programming. High level languages allow developers to code complex code that is then translated to machine language so that the computer can understand what needs to be done.

**Question: Is it true that Python can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java?**

*Answer:* Yes

**Question: What are different modes of programming in Python?**

*Answer:* There are two modes of programming in Python:

1. Interactive Mode Programming: In this, we invoke the interpreter without passing any script or python file. We can start the Python command line interpreter or the Python shell in IDLE and start passing instructions and get instant results.
2. Script Mode of Programming: Saving code in a Python file. When the script is executed the interpreter is invoked and it is active as long as the script is running. Once all the instructions of the script are executed the interpreter is no longer active.

**Question: Draw comparison between Java and Python.**

*Answer:*

1. Java is compiled and Python is interpreted: Both Java and Python are compiled down to the byte code on the virtual machine. However, in case of Python this happens automatically during runtime whereas Java has a separate program – javac to accomplish this task. This also means that if speed is the major concern in your project then Java may have an edge over Python but you cannot deny that Python allows faster development (as you will learn in the next point).
2. Java is Statistically typed and Python is dynamically typed: While coding in Python, it is not required to declare the type of the variable. This makes Python easy to write and read but it can be difficult to analyse. The developer is able to code faster and the task can be accomplished in fewer lines of code. Developing applications in Python can be much faster in Python than Java. However, Java's static type system makes it less prone to bugs.

3. Writing style:

Both Java and Python follow different writing styles. Java encloses everything in braces whereas Python follows indentation that makes the code neat and readable. Indentation also determines the code execution

4. Both Java and Python are efficient languages:

Both Java and Python are being widely used for web development frameworks. Developers are able to create complex applications that can handle high traffic.

5. Both Java and Python have strong community and library support:

Both are open source languages having communities that are offering support and contribution to the language. You can easily find a library for almost anything.

6. Python can be more budget friendly:

Since Python is dynamically typed, it makes development a fast process as a result of which developers can expect to develop an application in a record time thereby lowering down the cost of development.

7. Java is more preferred language in mobile development:

Android app development is mostly done using Java and XML. However, there are libraries like Kivy which can be used along with Python code in order to make it compatible for android development.

8. Python is preferred for Machine learning, Internet of things, ethical hacking, data analysis, and artificial intelligence:

Python has very specialized libraries and general flexibility and hence it has become number one choice for projects that have bias towards deep learning, machine learning, and image recognition.

9. Java and Python both support OOP:

10. LOC in Java is more than in Python:

Let's have a look at how we would go about printing simple *Hello World* in Java:

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
Where as in python we just write one line of code:
print("Hello World")
```

## 6 - Python Interview Questions

11. Java is more difficult to learn as compared to Python: Python was developed with main focus on making it easy to learn.
12. Java is stronger when it comes to connectivity with database. The Database access layer of Java is very strong and is compatible with almost any database. Python database connectivity is not as strong as Java.
13. Security: Java gives high priority to security which is why it is a preferred language for applications that require security but a good developer can code a secure application in Python also.

### **Question: Once Python is installed, how can we start working on code?**

*Answer:* After Python is installed there are three ways to start working on code:

1. You can start interactive interpreter from the command line and start writing the instructions after the >>> prompt.
2. If you intend to write a code of several lines then it would be a wise decision to save your file or script with .py extension and you can execute these files from command line. Multiline programs can be executed on an interactive interpreter, also but it does not save the work.
3. Python also has its own GUI environment known as Integrated Development Environment (IDLE). IDLE helps programmers write code faster as it helps with automatic indenting and highlights different keywords in different colors. It also provides an interactive environment. It has two windows: the shell provides interactive environment whereas the editor allows you to save your scripts before executing the code written in it.

### **Question: What is the function of interactive shell?**

*Answer:* The interactive shell stands between the commands given by the user and the execution done by the operating system. It allows users to use easy shell commands and the user need not be bothered about the complicated basic functions of the Operating System. This also protects the operating system from incorrect usage of system functions.

### **Question: How to exit interactive mode?**

*Answer:* `Ctrl+D` or `exit()` can be used to quit interactive mode.

**Question: Which character set does Python use?**

*Answer:* Python uses traditional ASCII character set.

**Question: What is the purpose of indentation in Python?**

*Answer:* Indentation is one of the most distinctive features of Python. While in other programming languages, developers use indentation to keep their code neat but in case of Python, indentation is required to mark the beginning of a block or to understand which block the code belongs to. No braces are used to mark blocks of code in Python. Blocks in code are required to define functions, conditional statements, or loops. These blocks are created simply by correct usage of spaces. All statements that are same distance from the right belong to the same block.

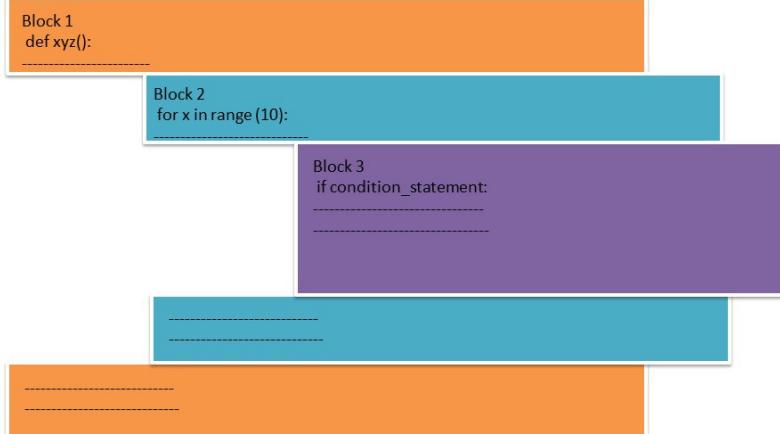


Figure 1

**Remember**

- The first line of the block always ends with a semicolon (:).
- Code under the first line of block is indented. The preceding diagram depicts a scenario of indented blocks.
- Developers generally use four spaces for the first level and eight spaces for a nested block, and so on.

**Question: Explain Memory Management in Python.**

*Answer:* Memory management is required so that partial or complete section of computer's memory can be reserved for executing programs and

## 8 - Python Interview Questions

processes. This method of providing memory is called memory allocation. Also, when data is no longer required, it must be removed. Knowledge of memory management helps developers develop efficient code.

Python makes use of its private heap space for memory management. All object structures in Python are located in this private heap (which is not accessible by the programmer). Python's memory manager ensures that this heap space is allocated judiciously to the objects and data structures. An in built garbage collector in Python recycles the unused memory so that it is available in heap space.

Everything in Python is an object. Python has different types of objects, such as simple objects which consist of numbers and strings and container objects such as dict, list, and user defined classes. These objects can be accessed by an identifier- name. Now, let's have a look at how the things work.

Suppose, we assign value of 5 to a variable a:

```
a = 5
```

Here, '5' is an integer object in memory and 'a' has reference to this integer object.

```
>>>  
>>> a = 5  
>>> id(a)  
140718128935888  
>>>
```

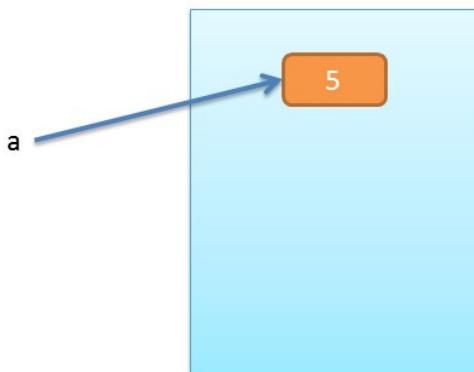


Figure 2

In the above illustration, the `id()` function provides a unique identification number of an object. This unique identification number is an integer value which will remain unique and constant for the object during its lifetime. Two objects with non-overlapping lifetimes can have the same `id()` value.

The id for integer object 5 is 140718128935888. Now we assign the same value 5 to variable b. You can see in the following diagram that both a and b have reference to the same object.

```
>>>
>>> a = 5
>>> id(a)
140718128935888
>>>
>>>
>>> b = 5
>>> id(b)
140718128935888
>>>
```

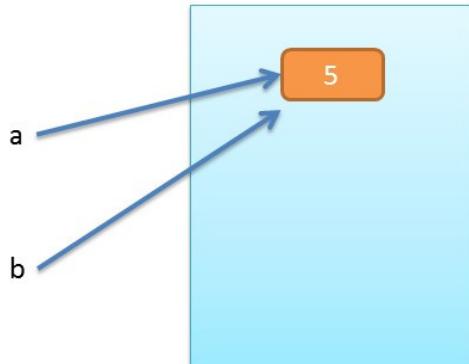


Figure 3

Now, let us say:

$c = b$ .

This means, c too will have reference to the same object.

```
>>>
>>> a = 5
>>> id(a)
140718128935888
>>>
>>>
>>> b = 5
>>> id(b)
140718128935888
>>>
>>> c = b
>>> id(c)
140718128935888
>>>
```

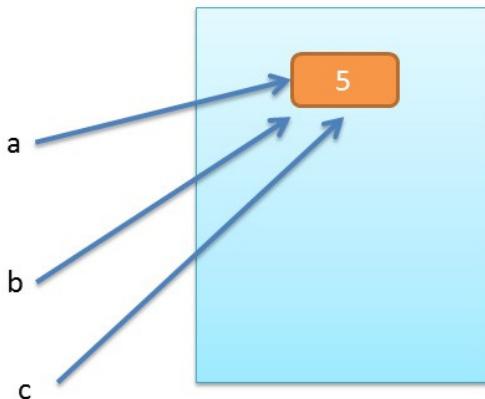


Figure 4

Now, suppose we perform the following operation:

$a = a + 1$

## 10 - Python Interview Questions

This means that `a` is now equal to 6 and now refers to a different object.

```
>>>
>>> a = 5
>>> id(a)
140718128935888
>>>
>>>
>>> b = 5
>>> id(b)
140718128935888
>>>
>>> c = b
>>> id(c)
140718128935888
>>>
>>>
>>> a = a+1
>>> a
6
>>> id(a)
140718128935920
>>>
```

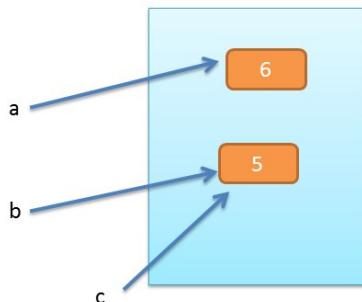


Figure 5

Some amount of memory organization is done for every instruction. The underlying operating system allocates some amount of memory for every operation. The Python interpreter gets its share of memory depending on various factors such as version, platform and environment.

The memory assigned to the interpreter is divided into the following:

1. Stack:

- a. Here all methods are executed.
- b. References to objects in the heap memory are created in stack memory.

2. Heap:

- a. The objects are created in Heap memory.

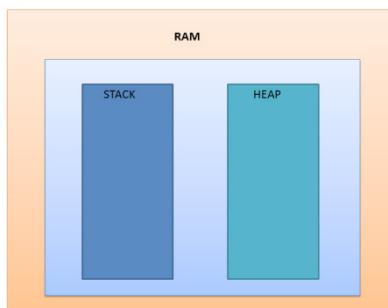


Figure 6

Now let's have a look at how the things work with the help of an example. Look at the following piece of the code:

```
def function1(x):
    value1 = (x + 5)* 2
    value2 = function2(value1)
    return value2
def function2(x):
    x = (x*10)+5
    return x
x = 5
final_value = function1(x)
print("Final value = ", final_value)
```

Now, let's see how this works. The program's execution starts from main which in this case is:

```
x = 5
final_value = function1(x)
print("Final value = ", final_value)
```

Step 1: execute `x = 5`

This creates integer object 5 in the heap and reference to this object i.e. `x` is created in the main stack memory.

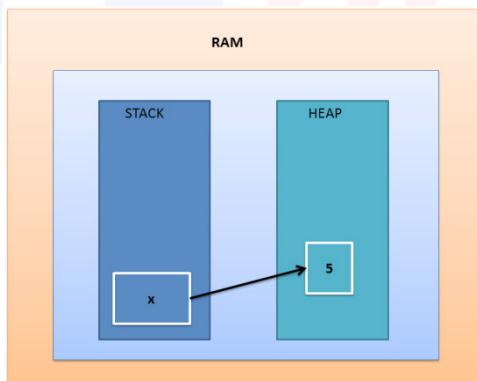


Figure 7

Step 2: is to execute `final_value = .function1(x)`

This statement calls `function1()`

## 12 - Python Interview Questions

```
def function1(x):
    value1 = (x + 5)* 2
    value2 = function2(value1)
    return value2
```

In order to execute function1() a new stack frame is added in the memory. Till the time function1() is being executed the lower stack frame of x referencing to 5 is put hold. The integer value 5 is passed as parameter to this function.

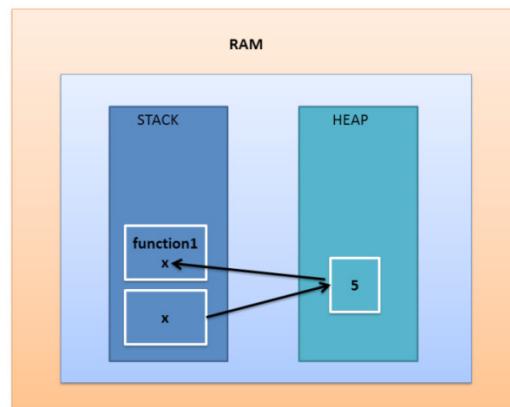


Figure 8

Now,  $value1 = (x+5)* 2 = (5+5)*2 = 10*2 = 20$

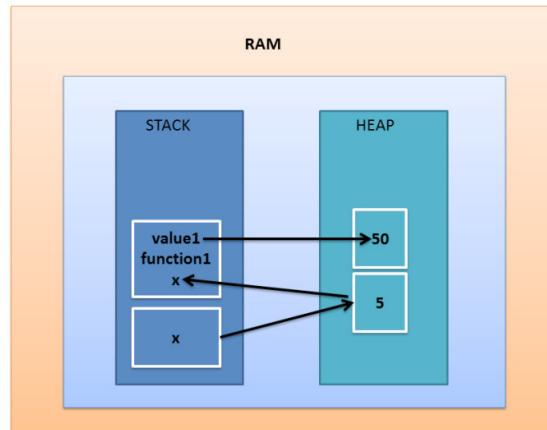


Figure 9

function1() assigns the value of 50 to value1.

The next step is: value2 = function2(value1)

Here function2() is called to evaluate a value that needs to be passed on to value2. In order to accomplish this, another memory stack is created. The integer object value1 having a value of 20 is passed as reference to function2.

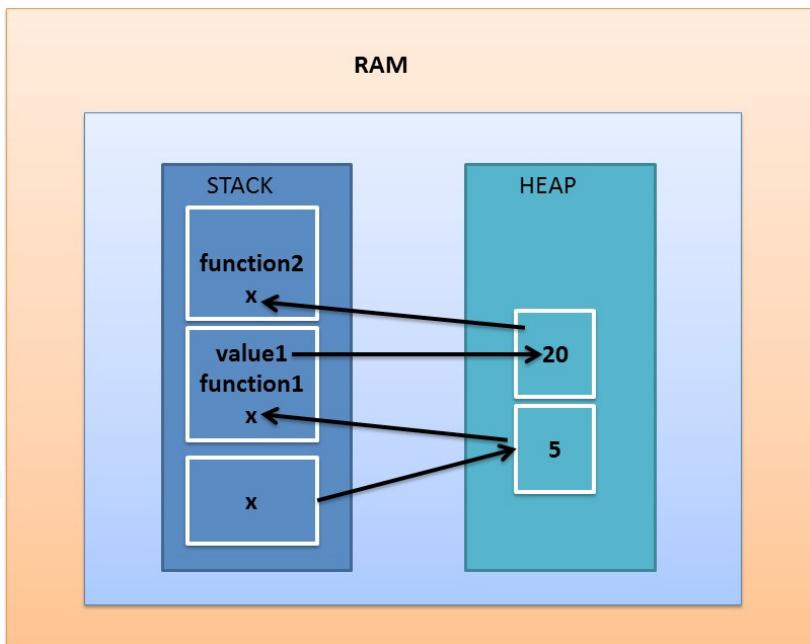


Figure 10

```
def function2(x):
    x = (x*10)+5
    return x
```

The function function2() evaluates the following expression and returns the value:

$$\begin{aligned}x &= (x*10)+5 \\x &= (20*10)+5 = (200)+5 = 205\end{aligned}$$

## 14 - Python Interview Questions

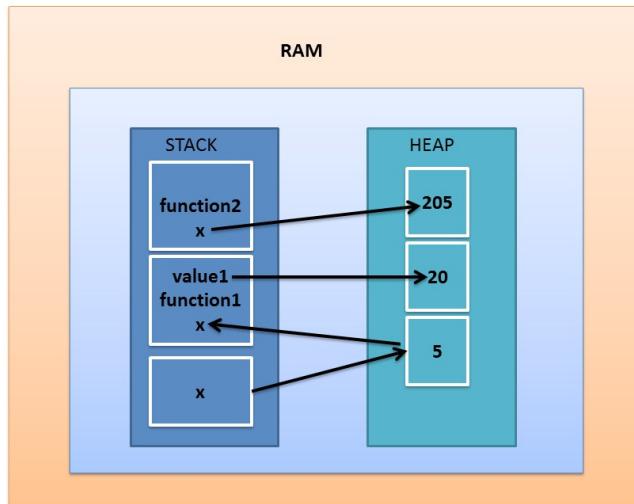


Figure 11

The function `function2()` is fully executed and value 205 is assigned to `value2` in function 1. So, now the stack for `function2` is removed.

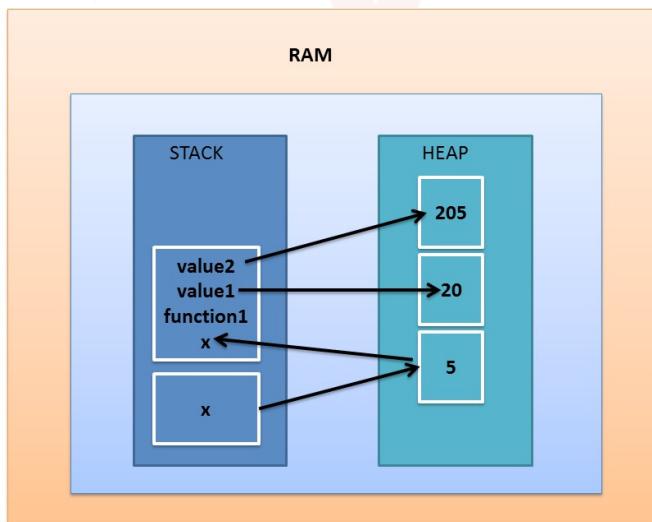


Figure 12

Now function1() will return the value 205 and it will be assigned to final\_value in main stack.

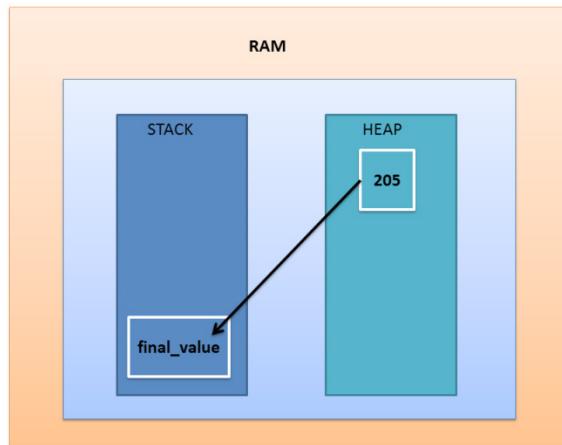


Figure 13

Here it is important to note that you would see that x exists in the main as well as in different functions but the values don't interfere with each other as each x is in a different memory stack.

**Question:** Explain Reference counting and Garbage collection in Python.

**Answer:** Unlike languages like C/ C++, the process of allocation and deallocation of memory in Python is automatic. This is achieved with the help of reference counting and garbage collection.

As the name suggests, reference counting counts the number of times an object is referred to by other objects in a program. Every time a reference to an object is eliminated, the reference count decreases by 1. As soon as the reference count becomes zero, the object is deallocated. An object's reference count decreases when an object is deleted, reference is reassigned or the object goes out of scope. The reference count increases when an object is assigned a name or placed in a container.

Garbage collection on the other hand allows Python to free and reclaim blocks of memory that are no longer of any use. This process is carried out periodically. Garbage collector runs while the program is being executed and the moment the reference count of an object reaches zero, the garbage collector is triggered.

## 16 - Python Interview Questions

### Question: What are multi-line statements?

Answer: All the statements in Python end with a newline. If there is a long statement, then it is a good idea to extend it over multiple lines. This can be achieved using the continuation character (\).

Explicit line continuation is when we try to split a statement into multiple lines where as in case of implicit line continuation we try to split parentheses, brackets, and braces into multiple lines.

Example for multiple line statements:

Explicit:

```
>>> first_num = 54
>>> second_num = 879
>>> third_num = 876
>>> total = first_num +\
second_num+\nthird_num
>>> total
1809
>>>
Implicit:
>>> weeks=['Sunday',
'Monday',
'Tuesday',
'Wednesday',
'Thursday',
'Friday',
'Saturday']
>>> weeks
['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']
>>>
```

### Question: What are the types of error messages in Python?

Answer: While working with Python you can come across:

1. Syntax-error
2. Run-time errors

Syntax errors are static errors that are encountered when the interpreter is reading the program. If there is any mistake in the code then the interpreter will display a syntax error message and the program will not be executed.

Run time errors as the name suggests are dynamic error. They are detected while the program is executing. Such errors are bugs in the program that may require design change such as running out of memory, dividing a number by zero, and so on.

**Question: What are the advantages of Python's IDLE environment?**

*Answer:* The Python IDLE environment has the following:

- Python's interactive mode
- Tools for writing and running programs
- It comes along with the text editors which can be used for working on scripts

**Question: What is a comment?**

*Answer:* A comment is one or more statements used to provide documentation or information about a piece of code in a program. In Python one line comments start with '#'.

**Question: Is Python a case - sensitive programming language?**

*Answer:* Yes

## Chapter 2

# Data Types and Their in-built Functions

Data types in Python

Numbers : int, float, and complex

List : Ordered sequence of items

Tuple : Ordered sequence of items similar to list but is immutable

Strings : Sequence of characters

Set : unordered collection of unique items

Dictionary : unordered collection of key-value pair

### **Question: Differentiate between mutable and immutable objects?**

*Answer:*

Mutable Objects	Immutable Objects
Can change their state or	Cannot change their state or contents
contents	
Type: list, dict, set	Inbuilt types : int, float, bool, string, unicode, tuple
Easy to change	Quick to access but making changes require creation of copy
Customized container like	Primitive like data types are immutable
types are mostly mutable	

- Mutable objects are recommended when one has the requirement to change the size or content of the object
- Exception to immutability: tuples are immutable but may contain elements that are mutable

### **Question: What is Variable in Python?**

*Answer:* Variables in Python are reserved memory locations that store values. Whenever a variable is created, some space is reserved in the

## 20 - Python Interview Questions

memory. Based on the data type of a variable, the interpreter will allocate memory and decide what should be stored in the memory.

```
>>> a = 9 #assign value to a
>>> type(a) #check type of variable a
<class 'int'>
>>>
```

**Question: How can we assign same value to multiple variables in one single go?**

**Answer:** Same value can be assigned to multiple variables in one single go as shown in the following code:

```
>>> a = b = c = "Hello World!!!"
>>> a
'Hello World!!!'
>>> b
'Hello World!!!'
>>> c
'Hello World!!!'
>>>
```

### Numbers

Types of numbers supported are as follows:

1. Signed integers int: these can be +ve or -ve and do not have any decimal point
2. Long integers long: these are integers of unlimited size followed by a capital or lowercase ‘L’
3. Float : are real numbers with decimal points
4. Complex numbers: have real and imaginary parts

**Question: What are the methods available for conversion of numbers from one type to another?**

**Answer:** The following functions are available to convert numbers from one form to another.

```
# convert to integer
a = 87.8
print("a = ",a)
print("*****")
```

```
#After conversion to integer
print("After conversion to integer value of a
will be a = ", int(a))
print("*****")
# convert to float
a = 87
print("a = ",a)
print("*****")
#After conversion to float
print("After conversion to float value of a will
be a = ", float(a))
print("*****")
# convert to complex
a = 87
print("a = ",a)
#After conversion to complex
print("After conversion to complex value of a
will be = ", complex(a))
print("*****")
```

### Output

```
a = 87.8
*****
After conversion to integer, value of a will be a
= 87
*****
a = 87
*****
After conversion to float value of a will be a =
87.0
*****
a = 87
After conversion to complex value of a will be =
(87+0j)
*****
>>>
```

**Question:** What are the mathematical functions defined in Python to work with numbers?

**Answer:** Mathematical functions are defined in math module. You will have to import this module in order to work with these functions.

## 22 - Python Interview Questions

```
import math
#ceiling value
a = -52.3
print ("math ceil for ",a, " is : ", math.
ceil(a))
print("*****")
#exponential value
a = 2
print("exponential value for ", a ,," is: ",math.
exp(2))
print("*****")
#absolute value of x
a = -98.4
print ("absolute value of ",a," is: ",abs(a))
print("*****")
#floor values
a = -98.4
print ("floor value for ",a," is: ",math.floor(a)) print("*****")
# log(x)
a = 10
print ("log value for ",a," is : ", math.log(a))
print("*****")
# log10(x)
a = 56
print ("log to the base 10 for ",a," is : ",math.
log10(a))
print("*****")
# to the power of
a = 2
b = 3
print (a," to the power of ",b," is : ",math.
pow(2,3))
print("*****")
# square root
a = 2
print("sqaure root")
print ("Square root of ",a," is : ", math.
sqrt(25))
print("*****")
```

## Output

```
math ceil for -52.3 is : -52
*****
exponential value for 2 is: 7.38905609893065
*****
absolute value of -98.4 is: 98.4
*****
floor value for -98.4 is: -99
*****
log value for 10 is : 2.302585092994046
*****
log to the base 10 for 56 is :
1.7481880270062005
*****
2 to the power of 3 is : 8.0
*****
sqaure root
Square root of 2 is : 5.0 *****
```

**Question:** What are the functions available to work with random numbers?

**Answer:** To work with random numbers you will have to import random module. The following functions can be used:

```
import random
#Random choice
print (" Working with Random Choice") seq=
[8,3,5,2,1,90,45,23,12,54]
print ("select randomly from ", seq," : ",random.
choice(seq))
print("*****")
#randomly select from a range
print ("randomly generate a number between 1 and
10 : ",random.randrange(1, 10))
print("*****")
#random()
print ("randomly display a float value between 0
and 1 : ",random.random())
print("* * * * *")
```

## 24 - Python Interview Questions

```
#shuffle elements of a list or a tuple seq=
[1,32,14,65,6,75]
print("shuffle ",seq,"to produce : ",random.
shuffle(seq))
#uniform function to generate a random float number
#between two numbers
print ("randomly display a float value between 65
and 71 : ",random.uniform(65,71))
```

### Output

```
Working with Random Choice
select randomly from [8, 3, 5, 2, 1, 90, 45, 23,
12, 54] : 2
*****
randomly generate a number between 1 and 10 : 8
*****
randomly display a float value between 0 and 1 :
0.3339711273144338
* * * * *
shuffle [1, 32, 14, 75, 65, 6] to produce : None
randomly display a float value between 65 and 71 :
65.9247420528493
```

### Question: What are the trigonometric functions defined in the math module?

Answer: Some of the trigonometric functions defined in math module are as follows:

```
import math
# calculate arc tangent in radians
print ("atan(0) : ",math.atan(0))
print("*****")
# cosine of x
print ("cos(90) : ",math.cos(0))
print("*****")
# calculate hypotenuse
print ("hypot(3,6) : ",math.hypot(3,6))
print("*****")
```

```
# calculates sine of x
print ("sin(0) : ", math.sin(0))
print("*****")
# calculates tangent of x
print ("tan(0) : ",math.tan(0))
print("*****")
# converts radians to degree
print ("degrees(0.45) : ",math.degrees(0.45))
print("*****")
# converts degrees to radians
print ("radians(0) : ",math.radians(0))
```

**Question: What are number data types in Python?**

*Answer:* Number data types are the one which are used to store numeric values such as:

1. integer
2. long
3. float
4. complex

Whenever you assign a number to a variable it becomes numeric data

```
>>> a = 1
type.
>>> b = -1
>>> c = 1.1
>>> type(a)
<class 'int'>
>>> type(b)
<class 'int'>
>>> type(c)
<class 'float'>
```

**Question: How will you convert float value 12.6 to integer value?**

*Answer:* Float value can be converted to integer value by calling int() function.

```
>>> a = 12.6
>>> type(a)
      <class
'float'>    >>>
int(a)
12
>>>
```

**Question: How can we delete reference to a variable?**

*Answer:* You can delete reference to an object using *del* keyword.

```
>>> a=5
>>> a
5
>>> del a
>>> a
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
a
NameError: name 'a' is not defined
>>>
```

**Question: How will you convert real numbers to complex numbers?**

*Answer:* Real numbers can be converted to complex numbers as shown in the following code:

```
>>> a = 7
>>> b = -8
>>> x = complex(a,b)

>>> x.real
7.0
>>> x.imag
-8.0
>>>
```

## Keywords, Identifiers, and Variables

### Keywords

- Keywords are also known as reserved words.

- These words cannot be used as name for any variable, class, or function.
- Keywords are all in lower case letters.
- Keywords form vocabulary in Python.
- Total number of keywords in Python are 33.
- Type `help()` in Python shell, a `help>` prompt will appear. Type `keywords`. This would display all the keywords for you. The list of keywords is highlighted for you.

```
Welcome to Python 3.7's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> keywords

Here is a list of the Python keywords. Enter any keyword to get more help.

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

help>
```

Figure 14

## Identifiers

- Python Identifier is a name given to variable, function, or a class.
- As the name suggests identifiers provide an identity to a variable, function, or a class.
- An identifier name can start with upper or lower case letters or an underscore followed by letters and digits.
- An identifier name cannot start with a digit.
- An identifier name can only contain letters, digits and an underscore.
- Special characters such as @, %, !, #, \$, .' cannot be a part of identifier name.

## 28 - Python Interview Questions

- As per naming convention, generally the class name starts with a capital letter and the rest of the identifiers in a program should start with lower case letters.
- If an identifier starts with a single underscore then it is private and two leading underscore in front of an identifier's name indicate that it is strongly private.
- Avoid using an underscore '\_' as leading or trailing character in an identifier because this notation is being followed for Python built-in types.
- If an identifier also has two trailing underscore then it is language defined special name.
- Though it is said that a Python identifier can be of unlimited length but having a name of more than 79 characters is violation of PEP-8 standard which asks to limit all line to a maximum of 79 characters. You can check if an identifier is valid or not by calling `iskeyword()` function as shown in the following code:

```
>>> import keyword  
>>> keyword.iskeyword("if")  
True  
>>> keyword.iskeyword("only")  
False
```

## Variables

- Variables are nothing but a label to a memory location that holds a value.
- As the name suggests, the value of a variable can change.
- You need not declare a variable in Python but they must be initialized before use E.g, `counter =0`.
- When we pass an instruction `counter=0`, it creates an object and a value 0 is assigned to it. If the counter variable already exists then it will be assigned a new value 0 and if it does not exists then it will get created.
- By assigning a value we establish an association between a variable and an object.
- `counter=0` means that the variable counter refers to a value of '0' in memory. If a new value is assigned to counter then that means the

variable now refers to a new memory chunk and the old value was garbage collected.

```
>>> counter = 0
>>> id(counter)
140720960549680
>>> counter = 10
>>> id(counter)
140720960550000
>>>
```

**Question: What are tokens?**

*Answer:* Tokens are the smallest units of program in Python. There are four types of tokens in Python:

- Keywords
- Identifiers
- Literals
- Operators

**Question: What are constants?**

*Answer:* Constants (literals) are values that do not change while executing a program.

**Question: What would be the output for  $2*4^{**}2$ ? Explain.**

*Answer:* The precedence of  $^{**}$  is higher than precedence of  $*$ . Thus,  $4^{**}2$  will be computed first. The output value is 32 because  $4^{**}2 = 16$  and  $2*16 = 32$ .

**Question: What would be the output for the following expression:**

```
print('{0:.4}'.format(7.0 / 3))
```

*Answer:* 2.333

**Question: What would be the output for the following expression?**

```
print('{0:.4%}'.format(1 / 3))
```

*Answer:* 33.3333%

**Question: What would be the value of the following expressions?**

1. ~5

2. ~~5
3. ~~~5

*Answer:*  $\sim x = -(x+1)$ . Therefore the output for the given expressions would be as follows:

1. -6
2. 5
3. -6

We will now have a look at three most important sequence types in Python. All three represent collection of values which are placed in order. These three types are as follows:

1. String: immutable sequence of text characters. There is no special class for a single character in Python. A character can be considered as String of text having a length of 1.
2. List: Lists are very widely used in Python programming and a list represents sequence of arbitrary objects. List is mutable.
3. Tuple: Tuple is more or less like a list but it is immutable.

## Strings

- Sequence of characters.
- Once defined cannot be changed or updated hence strings are immutable.
- Methods such as replace(), join() etc. can be used to modify a string variable. However, when we use these methods, the original string is not modified instead Python creates a copy of the string which is modified and returned.

### Question: How can String literals be defined?

*Answer:* Strings can be created with single/double/triple quotes.

```
>>> a = "Hello World"  
>>> b = 'Hi'  
>>> type(a)  
<class 'str'>  
>>> type(b)  
<class 'str'>
```

```
>>>
>>> c = "Once upon a time
in a land far far away
there lived a king"
>>> type(c)
<class 'str'>
>>>
```

### Question: How can we perform concatenation of Strings?

Answer: Concatenation of Strings can be performed using following techniques:

#### 1. + operator

```
>>> string1 = "Welcome"
>>> string2 = " to the world of Python!!!"
>>> string3 = string1 + string2
>>> string3
'Welcome to the world of Python!!!'
>>>
```

#### 2. Join() function

The join() function is used to return a string that has string elements joined by a separator. The syntax for using join() function.

```
string_name.join(sequence)
>>> string1 = "-"
>>> sequence = ("1","2","3","4")
>>> print(string1.join(sequence))
1-2-3-4
>>>
```

#### 3. % operator

```
>>> string1 = "HI"
>>> string2 = "THERE"
>>> string3 = "%s %s" %(string1, string2)
>>> string3
'HI THERE'
>>>
```

#### 4. format() function

```
>>> string1 = "HI"
>>> string2 = "THERE"
```

## 32 - Python Interview Questions

```
>>> string3 = "{} {}".format(string1, string2)
>>> string3
'HI THERE'
>>>
```

### 5. f-string

```
>>> string1 = "HI"
>>> string2 = "THERE"
>>> string3 = f'{string1} {string2}'
>>> string3
'HI THERE'
>>>
```

### Question: How can you repeat strings in Python?

Answer: Strings can be repeated either using the multiplication sign “\*” or by using for loop.

- operator for repeating strings

```
>>> string1 = "Happy Birthday!!!"
>>> string1*3
'Happy Birthday!!!Happy Birthday!!!Happy
Birthday!!!'
>>>
```

- for loop for string repetition

```
for x in range(0,3)

>>> for x in range(0,3):
print("HAPPY BIRTHDAY!!!")
```

### Question: What would be the output for the following lines of code?

```
>>> string1 = "HAPPY "
>>> string2 = "BIRTHDAY!!!"
>>> (string1 + string2)*3
```

Answer:

```
'HAPPY BIRTHDAY!!!HAPPY BIRTHDAY!!!HAPPY
BIRTHDAY!!!'
```

**Question:** What is the simplest way of unpacking single characters from string “HAPPY”?

**Answer:** This can be done as shown in the following code:

```
>>> string1 = "HAPPY"
>>> a,b,c,d,e = string1
>>> a
'H'
>>> b
'A'
>>> c
'P'
>>> d
'P'
>>> e
'Y'
>>>
```

**Question:** Look at the following piece of code:

```
>>> string1 = "HAPPY"
>>> a,b = string1
```

What would be the outcome for this?

**Answer:** This code will generate error stating *too many values to unpack* because the number of variables do not match the number of characters in the strings.

**Question:** How can you access the fourth character of the string “HAPPY”?

**Answer:** You can access any character of a string by using Python’s array-like indexing syntax. The first item has an index of 0. Therefore, the index of fourth item will be 3.

```
>>> string1 = "HAPPY"
>>> string1[3]
'P'
```

**Question:** If you want to start counting the characters of the string from the right most end, what index value will you use (assuming that you don’t know the length of the string)?

## 34 - Python Interview Questions

**Answer:** If the length of the string is not known we can still access the rightmost character of the string using index of -1.

```
>>> string1 = "Hello World!!!"  
>>> string1[-1]  
'!'  
>>>
```

**Question:** By mistake the programmer has created string *string1* having the value “HAPPU”. He wants to change the value of the last character. How can that be done?

**Answer:** Strings are immutable which means that once they are created they cannot be modified. If you try to modify the string it will generate an error.

```
>>> string1 = "HAPPU"  
>>> string1[-1] = "Y"  
Traceback (most recent call last):  
File "<pyshell#9>", line 1, in <module>  
    string1[-1] = "Y"  
TypeError: 'str' object does not support item assignment
```

However, there is a way out for this problem. We can use the *replace()* function.

```
>>> string1 = "HAPPU"  
>>> string1.replace('U', 'Y')  
'HAPPY'
```

Here, in case of *replace()* function, a new string is created and the value is reassigned to *string1*. So, *string1* is not modified but is actually replaced.

**Question: Which character of the string will exist at index -2?**

**Answer:** Index of -2 will provide second last character of the string:

```
>>> string1 = "HAPPY"  
>>> string1[-1]  
'Y'  
>>> string1[-2]  
'P'  
>>>
```

**Question:** >>> str1 = "\t\tHi\n"

```
>>> print(str1.strip())
```

What will be the output?

*Answer:* Hi

**Question:** Explain slicing in strings.

*Answer:* Python allows you to extract a chunk of characters from a string if you know the position and size. All we need to do is to specify the start and end point. The following example shows how this can be done. In this case we try to retrieve a chunk of string starting at index 4 and ending at index 7. The character at index 7 is not included.

```
>>> string1 = "HAPPY-BIRTHDAY!!!"  
>>> string1[4:7]  
'Y-B'  
>>>
```

If, in the above example you omit the first index, then default value of 0 is considered and slicing of text chunk starts from the beginning of the string.

```
>>> string1 = "HAPPY-BIRTHDAY!!!"  
>>> string1[:7]  
'HAPPY-B'  
>>>
```

Same way if you don't mention the second index then the chunk will be taken from the starting position till the end of the string.

```
>>> string1 = "HAPPY-BIRTHDAY!!!"  
>>> string1[4:]  
'Y-BIRTHDAY!!!'
```

Value of string1[:n]+string1[n:] will always be the same string.

```
>>> string1[:4]+ string1[4:]  
'HAPPY-BIRTHDAY!!!'
```

Negative index can also be used with slicing but in that case the counting would begin from end.

```
>>> string1 = "HAPPY-BIRTHDAY!!!"  
>>> string1[-5:-1]  
'AY!!'  
>>>
```

## 36 - Python Interview Questions

You can also provide three index values:

```
>>> string1[1:7:2]
'AP-'
>>> string1[1:9:3]
'AYI'
>>>
```

Here, the first index is starting point, second index is ending point and character is not included and the third index is the stride or how many characters you would skip before retrieving the next character.

**Question: What would be the output for the following code?**

```
>>> string1 = "HAPPY-BIRTHDAY!!!"
>>> string1[-1:-9:-2]
```

Answer: '!!AH'

**Question: How does the *split()* function work with strings?**

: Answer We can retrieve chunk of string based on the delimiters that we provide. The split() operation returns the substrings without the delimiters.

Example 1

```
>>> string1 = "Happy Birthday"
>>> string1.split()
['Happy', 'Birthday']
```

Example 2

```
>>> time_string = "17:06:56"
>>> hr_str,min_str,sec_str = time_string.
split(":")
>>> hr_str
'17'
>>> min_str
'06'
>>> sec_str
'56'
>>>
```

You can also specify, how many number of times you want to split the string.

```
>>> date_string = "MM-DD-YYYY"
>>> date_string.split("-",1)
['MM', 'DD-YYYY']
>>>
```

In case you want Python to look for delimiters from the end and then split the string then you can use rsplit() method.

```
>>> date_string = "MM-DD-YYYY"
>>> date_string.rsplit("-",1)
['MM-DD', 'YYYY']
>>>
```

**Question:** What is the difference between the *split()* and *partition()* function?

*Answer:* The result of a partition function is a tuple and it retains the delimiter.

```
>>> date_string = "MM-DD-YYYY"
>>> date_string.partition("-")
('MM', ' ', 'DD-YYYY')
```

The rpartition() function on the other hand looks for the delimiter from the other end.

```
>>> date_string = "MM-DD-YYYY"
>>> date_string.rpartition("-")
('MM-DD', ' ', 'YYYY')
>>>
```

**Question:** Name the important escape sequence in Python.

*Answer :* Some of the important escape sequences in Python are as follows:

- \\: Backslash
- \': Single quote
- \": Double quote
- \f: ASCII form feed
- \n: ASCII linefeed
- \t: ASCII tab
- \v: Vertical tab

## String Methods

### capitalize()

Will return a string that has first alphabet capital and the rest are in lower case.

```
>>> string1 = "HAPPY BIRTHDAY"  
>>> string1.capitalize()  
'Happy birthday'  
>>>
```

### casefold()

Removes case distinction in string. Often used for caseless matching.

```
>>> string1 = "HAPPY BIRTHDAY"  
>>> string1.casefold()  
'happy birthday'  
>>>
```

### centre()

The function takes two arguments: (1) length of the string with padded characters (2) padding character (this parameter is optional)

```
>>> string1 = "HAPPY BIRTHDAY"  
>>> new_string = string1.center(24)  
>>> print("Centered String: ", new_string)  
Centered String: HAPPY BIRTHDAY  
>>>
```

### count()

Counts the number of times a substring repeats itself in the string.

```
>>> string1 = "HAPPY BIRTHDAY"  
>>> string1.count("P")  
2  
>>>
```

You can also provide the start index and ending index within the string where search ends.

### encode()

Allows you to encode unicoded strings to encodings supported by python.

```
>>> string1 = "HAPPY BIRTHDAY"  
>>> string1.encode()  
b'HAPPY BIRTHDAY'
```

By default python uses utf-8 encoding.

It can take two parameters:

encoding – type a string has to be encoded to and

error- response when encoding fails

```
>>> string1 = "HAPPY BIRTHDAY"  
>>> string1.endswith('Y')  
True  
>>> string1.endswith('i')  
False
```

endswith()

It returns true if a string ends with the substring provided else it will return false.

```
>>> string1 = "to be or not to be"  
>>> string1.endswith("not to be")  
True  
>>>
```

find()

Get the index of the first occurrence of a substring in the given string.

```
>>> string1 = "to be or not to be"  
>>> string1.find('be')  
3  
>>>
```

format()

The format function allows you to substitute multiple values in a string. With the help of positional formatting we can insert values within a string. The string must contain {} braces, these braces work as placeholder. This is where the values will be inserted. The format() function will insert the values.

Example 1:

```
>>> print("Happy Birthday {}".format("Alex"))  
Happy Birthday Alex
```

Example 2:

```
>>> print("Happy Birthday {}, have a {} day!!".  
format("Alex","Great"))  
Happy Birthday Alex, have a Great day!!  
>>>
```

## 40 - Python Interview Questions

Values that exist in the format() are tuple data types. A value can be called by referring to its index value.

Example 3:

```
>>> print("Happy Birthday {0}, have a {1} day!!".
format("Alex","Great"))
Happy Birthday Alex, have a Great day!!
>>>
```

More type of data can be added to the code using {index: conversion} format where index is the index number of the argument and conversion is conversion code of data type.

s- string

d- decimal

f – float

c- character

b- binary

o- octal

x- hexadecimal with lower case letters after 9

X-hexadecimal with upper case letters after 9

e- exponent

Example 4:

```
>>> print("I scored {0:.2f}% in my exams".
format(86))
```

I scored 86.00% in my exams.

`index()`

It provides the position of first occurrence of the substring in the given string.

```
>>> string1 = "to be or not to be"
>>> string1.index('not')
9
>>>
```

`isalnum()`

It returns true if a string is alphanumeric else it returns false.

```
>>> string1 = "12321$%%^&*"
>>> string1.isalnum()
```

```
False
>>> string1 = "string1"
>>> string1.isalnum()
True
>>>

isalpha()
```

It returns true if the whole string has characters or returns false.

```
>>> string1.isalpha()
False
>>> string1 = "tobeornottobe"
>>> string1.isalpha()
True
>>>
```

isdecimal()

Returns true if the string has all decimal characters.

```
>>> string1 = "874873"
>>> string1.isdecimal()
True
```

isdigit()

Returns true if all the characters of a string are digits:

```
>>> string1 = "874a873"
>>> string1.isdigit()
False
>>>
```

islower()

```
>>> string1 = "tIger"
>>> string1.islower()
False
>>>
```

isnumeric()

Returns true if the string is not empty characters of a string are numeric.

```
>>> string1 = "45.6"
>>> string1.isnumeric()
False
>>>
```

## 42 - Python Interview Questions

`isspace()`

Returns true if the string only has space.

```
>>> string1 = " "
>>> string1.isspace()
True
>>>
```

`lower()`

Converts upper case letters to lower case.

```
>>> string1 ="TIGER"
>>> string1.lower()
'tiger'
>>>
```

`swapcase()`

Changes lower case letters in a string to upper case and vice-versa.

```
>>> string1 = "tIger"
>>> string1 = "tIger".swapcase()
>>> string1
'TIGER'
>>>
```

### Question: What are execution or escape sequence characters?

**Answer:** Characters such as alphabets, numbers or special characters can be printed easily. However, whitespaces such as line feed, tab, etc. cannot be displayed like other characters. In order to embed these characters we have used execution characters. These characters start with a backslash character (\) followed by a character as shown in the following code:

1. \n stands for end of line.

```
>>> print("Happy \nBirthday")
Happy
```

Birthday

2. \\ prints backslash - \'

```
>>> print('\\')
\
```

3. \t prints a tab

```
>>> print("Happy\tBirthday")
Happy Birthday
```

## Lists

- Σ Lists are ordered and changeable collection of elements.  
Can be created by placing all elements inside a square bracket.
- Σ All elements inside the list must be separated by comma “,”  
It can have any number of elements and the elements need not be of
- Σ same type.  
If a list is a referential structure which means that it actually stores
- Σ references to the elements.  
Lists are zero indexed so if the length of a string is “n” then the first element will have the index of 0 and the last element will have an index of n-1.
- Σ Lists are widely used in Python programming.  
Lists are mutable therefore they can be modified after creation.

### Question: What is a list?

*Answer:* A list is an in built Python data structure that can be changed. It is an ordered sequence of elements and every element inside the list may also be called an item. By ordered sequence, it is meant that every element of the list can be called individually by its index number. The elements of a list are enclosed in square brackets[].

```
>>> # create empty list
>>> list1 = []
>>> # create a list with few elements
>>> list2 = [12,"apple", 90.6,]
>>> list1
[]
>>> list2
[12, 'apple', 90.6]
>>>
```

### Question: How would you access the 3rd element of the following list?

```
list1 = ["h","e","l","p"]
```

What would happen when you try to access list1[4]?

*Answer:* The third element of list1 will have an index of 2. Therefore, we can access it in the following manner:

## 44 - Python Interview Questions

```
>>> list1 = ["h","e","l","p"]
>>> list1[2]
'l'
>>>
```

There are four elements in the list. Therefore, the last element has an index of 3. There is no element at index 4. On trying to access list1[4] you will get an “*IndexError: list index out of range*”

**Question:** list1 = ["h","e","l","p"]. What would be the output for list1[-2] and list1[-5]?

*Answer:* list1[-2] = 'l'

list1[-5] will generate *IndexError: list index out of range*

Similar to Strings, the slice operator can also be used on list. So, if the range given is [a:b]. It would mean that all elements from index a to index b will be returned. [a:b:c] would mean all the elements from index a to index b, with stride c.

**Question:** list1 = ["I","L","O","V","E","P","Y","T","H","O","N"]

What would be the value for the following?

1. list1[5]
2. list1[-5]
3. list1[3:6]
4. list1[:-3]
5. list1[-3:]
6. list1[:]
7. list1[1:8:2]
8. list1[::-2]
9. list1[4::3]

*Answer:*

```
>>> list1 = ["I","L","O","V","E","P","Y","T","H","O","N"]
>>> list1[5]
'P'
>>> list1[-5]
'Y'
>>> list1[3:6]
['V', 'E', 'P']
>>> list1[:-3]
['I', 'L', 'O', 'V', 'E', 'P', 'Y', 'T']
>>> list1[-3:]
['H', 'O', 'N']
>>> list1[:]
['I', 'L', 'O', 'V', 'E', 'P', 'Y', 'T', 'H', 'O',
 'N']
>>> list1[1:8:2]
['L', 'V', 'P', 'T']
>>> list1[::-2]
['I', 'O', 'E', 'Y', 'H', 'N']
>>> list1[4::3]
['E', 'T', 'N']
>>>
```

**Question:** list1 = ["I","L","O","V","E","P","Y","T","H","O","N"]

list2 = ['O','N','L','Y'] Concatenate

the two strings.

**Answer:**

```
>>> list1 = list1+list2
>>> list1
['I', 'L', 'O', 'V', 'E', 'P', 'Y', 'T', 'H', 'O',
 'N', 'O', 'N', 'L', 'Y']
>>>
```

**Question:** How can you change or update a value of element in a list?

**Answer:** You can change the value of an element with the help of assignment operator (=).

## 46 - Python Interview Questions

```
>>> list1 = [1,2, 78,45,93,56,34,23,12,98,70]
>>> list1 = [1,2,78,45,93,56,34,23,12,98,70]
>>> list1[3]
45
>>> list1[3]=67
>>> list1[3]
67
>>> list1
[1, 2, 78, 67, 93, 56, 34, 23, 12, 98, 70]
>>>
```

**Question: list1 = [1,2,78,45,93,56,34,23,12,98,70]**

list1[6:9]=[2,2,2]

What is the new value of list1?

*Answer:* [1, 2, 78, 45, 93, 56, 2, 2, 2, 98, 70]

**Question: What is the difference between append() and extend() function for lists?**

*Answer:* The append() function allows you to add one element to a list whereas extend() allows you to add more than one element to the list.

```
>>> list1 = [1,2, 78,45,93,56,34,23,12,98,70]
>>> list1.append(65)
>>> list1
[1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]
```

```
>>> list1.extend([-3,-5,-7,-5])
>>> list1
[1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65, -3,
-5, -7, -5]
>>>
```

**Question: list1 = ["h","e","l","p"]**

What would be the value of list1\*2?

*Answer:* ['h', 'e', 'l', 'p', 'h', 'e', 'l', 'p']

**Question: list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]**

list1 +=[87]

What is the value of list1?

Answer: [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65, 87]

**Question: list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]**

list1 =[65]

What will be the output?

Answer: The output will be TypeError: unsupported operand type(s) for -=: 'list' and 'list'

**Question: list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]**

**How will you delete second element from the list?**

Answer: An element can be deleted from a list using the 'del' keyword.

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> del(list1[1])  
>>> list1  
[1, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]  
>>>
```

**Question: list1 = [[1,4,5,9],[2,4,7,8,1]]**

List1 has two elements, both are lists.

How will you delete the first element of the second list contained in list1?

Answer:

```
>>> list1 = [[1,4,5,9],[2,4,7,8,1]]  
>>> del(list1[1][0])  
>>> list1  
[[1, 4, 5, 9], [4, 7, 8, 1]]  
>>>
```

**Question: How would you find length of a list?**

Answer: The len() function is used to find the length of a string.

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> len(list1)  
12  
>>>
```

**Question: list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]**

Insert a value 86 at 5th position.

## 48 - Python Interview Questions

*Answer:*

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> list1.insert(4,86)  
>>> list1  
[1, 2, 78, 45, 86, 93, 56, 34, 23, 12, 98, 70, 65]  
>>>
```

**Question: list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]**

**Remove the value 78 from list1.**

*Answer:* A specific element can be removed from a list by providing the value to remove() function.

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> list1.remove(78)  
>>> list1  
[1, 2, 45, 93, 56, 34, 23, 12, 98, 70, 65]  
>>>
```

**Question: What does the pop() function do?**

*Answer:* The pop() function can be used to remove an element from a particular index and if no index value is provided, it will remove the last element. The function returns the value of the element removed.

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> list1.pop(3)  
45  
>>> list1  
[1, 2, 78, 93, 56, 34, 23, 12, 98, 70, 65]  
>>> list1.pop()  
65  
>>> list1  
[1, 2, 78, 93, 56, 34, 23, 12, 98, 70]  
>>>
```

**Question: Is there a method to clear the contents of a list?**

*Answer:* Contents of a list can be cleared using clear() function.

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> list1.clear()  
>>> list1  
[]  
>>>
```

**Question:** list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65]

Find the index of element having value 93.

**Answer:** We can find the index of a value using the index() function.

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> list1.index(93)  
4  
>>>
```

**Question:** Write code to sort list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65].

**Answer:** We can sort a list using sort() function.

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> list1.sort()  
>>> list1  
[1, 2, 12, 23, 34, 45, 56, 65, 70, 78, 93, 98]  
>>>
```

**Question:** Reverse elements of list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65].

**Answer:**

```
>>> list1 = [1, 2, 78, 45, 93, 56, 34, 23, 12, 98,  
70, 65]  
>>> list1.reverse()  
>>> list1  
[65, 70, 98, 12, 23, 34, 56, 93, 45, 78, 2, 1]  
>>>
```

**Question:** How can we check if an element exists in a list or not?

**Answer:** We can check if an element exists in a list or not by using 'in' keyword:

## 50 - Python Interview Questions

```
>>> list1 = [(1, 2, 4), [18, 5, 4, 6, 2], [4, 6, 5, 7], 9, 23, [98, 56]]  
>>> 6 in list1  
False  
>>> member = [4,6,5,7]  
>>> member in list1  
True  
>>>
```

### Tuples

- Tuples are sequences just like lists but are immutable.
- Cannot be modified.
- Tuples may or may not be delimited by parenthesis().
- Elements in a tuple are separated by a comma. If a tuple has only one element then a comma must be placed after that element. Without a trailing comma a single value in simple parenthesis would not be considered as a tuple.
- Both Tuples and lists can be used under the same situations.

#### **Question: When would you prefer to use a tuple or a list?**

**Answer:** Tuples and lists can be used for similar situations but tuples are generally preferred for collection of heterogeneous data types whereas lists are considered for homogeneous data types. Iterating through a tuple is faster than iterating through list. Tuples are ideal for storing values that you don't want to change. Since Tuples are immutable, the values within are write-protected.

#### **Question: How can you create a Tuple?**

**Answer:** A Tuple can be created in any of the following ways:

```
>>> tup1 =()  
>>> tup2=(4,)  
>>> tup3 = 9,8,6,5  
>>> tup4= (7,9,5,4,3)  
>>> type(tup1)  
<class 'tuple'>  
>>> type(tup2)  
<class 'tuple'>  
>>> type(tup3)  
<class 'tuple'>  
>>> type(tup4)  
<class 'tuple'>
```

However, as mentioned before, the following is not a case of a tuple:

```
>>> tup5= (0)
>>> type(tup5)
<class 'int'>
>>>
```

**Question: tup1 = (1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65)**

How would you retrieve the 7th element of this tuple?

*Answer:* Accessing an element of a Tuple is same as accessing an element of a list.

```
>>> tup1 = (1, 2, 78, 45, 93, 56, 34, 23, 12, 98,
70, 65)
>>> tup1[6]
34
>>>
```

**Question: tup1 = (1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65)**

What would happen when we pass an instruction tup1[6]=6?

*Answer:* A tuple is immutable hence tup1[6]=6 would generate an error.

Traceback (most recent call last):

```
File "<pyshell#18>", line 1, in <module>
tup1[6]=6
TypeError: 'tuple' object does not support item
assignment
>>>
```

**Question: tup1 = (1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65)**

What would happen if we try to access the 6th element using tup1[5.0]?

*Answer:* The index value should always be an integer value and not a float. This would generate a type error. TypeError: 'tuple' object does not support item assignment.

**Question: tup1 = (1,2,4), [8,5,4,6],(4,6,5,7),9,23,[98,56]**

**What would be the value of tup1[1][0]?**

*Answer:* 8

**Question: tup1 = (1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65)**

**What is the value of tup1[-7] and tup1[-15]?**

*Answer:*

```
>>> tup1[-7]
56
>>> tup1[-15]
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
tup1[-15]
IndexError: tuple index out of range
>>>
```

**Question: tup1 = (1,2,4), [8,5,4,6],(4,6,5,7),9,23,[98,56]**

tup2 = (1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65)

Find the value of:

1. tup1[2:9]
2. tup2[:-1]
3. tup2[-1:]
4. tup1[3:9:2]
5. tup2[3:9:2]

*Answer:*

1. tup1[2:9]  
((4, 6, 5, 7), 9, 23, [98, 56])
2. tup2[:-1]  
(1, 2, 78, 45, 93, 56, 34, 23, 12, 98,
3. tup2[-1:]  
(70)
4. (65,)  
tup1[3:9:2]
5. (9, [98, 56])  
tup2[3:9:2]  
(45, 56, 23)

**Question: tup1 = (1,2,4), [8,5,4,6],(4,6,5,7),9,23,[98,56]**

What will happen if tup1[1][0]=18 instruction is passed? What would happen if we pass an instruction tup1[1].append(2)?

*Answer:* tup1[1] is a list object and list is mutable.

Tuple is immutable but if a element of a tuple is mutable then its nested elements can be changed.

```
>>> tup1 = (1,2,4), [8,5,4,6],(4,6,5,7),9,23,[98,5  
6]  
>>> tup1[1][0]=18  
>>> tup1[1]  
[18, 5, 4, 6]  
>>> tup1[1].append(2)  
>>> tup1  
((1, 2, 4), [18, 5, 4, 6, 2], (4, 6, 5, 7), 9, 23,  
[98, 56])  
>>>
```

**Question: tup1 = (1,2,4), [8,5,4,6],(4,6,5,7),9,23,[98,56]**

tup2 =(1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70, 65)

What would be the output for tup1+tup2?

*Answer:*

```
>>> tup1 = (1,2,4), [8,5,4,6],(4,6,5,7),9,23,[98,56]  
>>> tup2 =(1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70,  
65)  
>>> tup1 +=tup2  
>>> tup1  
((1, 2, 4), [18, 5, 4, 6, 2], (4, 6, 5, 7), 9, 23,  
[98, 56], 1, 2, 78, 45, 93, 56, 34, 23, 12, 98, 70,  
65)  
>>>
```

**Question: How can we delete a tuple?**

*Answer:* A tuple can be deleted using del command.

```
>>> tup1 = (1,2,4), [8,5,4,6],(4,6,5,7),9,23,[98,56]  
>>> del tup1  
>>> tup1  
Traceback (most recent call last):  
File "<pyshell#23>", line 1, in <module>  
    tup1  
NameError: name 'tup1' is not defined  
>>>
```

**Question:** `tup1 = ((1, 2, 4), [18, 5, 4, 6, 2], (4, 6, 5, 7), 9, 23, [98, 56])`

`tup2 = 1,6,5,3,6,4,8,30,3,5,6,45,98` What is  
the value of:

1. `tup1.count(6)`
2. `tup2.count(6)`
3. `tup1.index(6)`
4. `tup2.index(6)?`

*Answer:*

1. `tup1.count(6)`  
0
2. `tup2.count(6)`  
3
3. `tup1.index(6)`  
0
4. `tup2.index(6)`  
1

**Question:** How can we test if an element exist in a tuple or not?

*Answer:* We can check if an element exists in a tuple or not by using in keyword:

```
>>> tup1 = ((1, 2, 4), [18, 5, 4, 6, 2], (4, 6, 5,  
7), 9, 23, [98, 56])  
>>> 6 in tup1  
False  
>>> member = [4,6,5,7]  
>>> member in tup1  
False  
>>> member2 = (4, 6, 5, 7)  
>>> member2 in tup1  
True  
>>>.
```

**Question:** How can we get the largest and the smallest value in tuple?

*Answer:* We can use `max()` function to get the largest value and `min()` function to get the smallest value.

```
>>> tup1 = (4, 6, 5, 7)
>>> max(tup1)
7
>>> min(tup1)
4
>>>
```

**Question: How to sort all the elements of a tuple?**

*Answer:*

```
>>> tup1 =(1,5,3,7,2,6,8,9,5,0,3,4,6,8)
>>> sorted(tup1)
[0, 1, 2, 3, 3, 4, 5, 5, 6, 6, 7, 8, 8, 9]
```

**Question: How can we find sum of all the elements in a tuple?**

*Answer:* We can find sum of all elements by using `sum()` function.

```
>>> tup1 =(1,5,3,7,2,6,8,9,5,0,3,4,6,8)
>>> sum(tup1)
67
```

Enumerate function for tuple and list in for loop section mentioned.

Dictionary

- Unordered sets of objects.
- Also known as maps, hashmaps, lookup tables, or associative array.
- Data exists in key-value pair. Elements in a dictionary have a key and a corresponding value. The key and the value are separated from each other with a colon ':' and all elements are separated by comma.
- Elements of a dictionary are accessed by the "key" and not by index. Hence it is more or less like an associative array where every key is associated with a value and elements exists in an unordered fashion as key-value pair.
- Dictionary literals use curly brackets '{}'.

**Question: How can we create a dictionary object?**

*Answer:*

A dictionary object can be created in any of the following ways:

## 56 - Python Interview Questions

```
>>> dict1 = {}
>>> type(dict1)
<class 'dict'>

>>> dict2 = {'key1': 'value1', 'key2': 'value2',
'key3': 'value3', 'key4': 'value4'}
>>> dict2
{'key1': 'value1', 'key2': 'value2', 'key3':
'value3', 'key4': 'value4'}
>>> type(dict2)
<class 'dict'>

>>> dict3 = dict({'key1': 'value1', 'key2': 'value2',
'key3': 'value3', 'key4': 'value4'})
>>> dict3
{'key1': 'value1', 'key2': 'value2', 'key3':
'value3', 'key4': 'value4'}
>>> type(dict3)
<class 'dict'>

>>> dict4 = dict([('key1', 'value1'), ('key2',
'value2'), ('key3', 'value3'), ('key4', 'value4')])
>>> dict4
{'key1': 'value1', 'key2': 'value2', 'key3':
'value3', 'key4': 'value4'}
>>> type(dict4)
<class 'dict'>
```

### Question: How can you access values of a dictionary?

Answer: Values of a dictionary can be accessed by the help of the keys as shown in the following code:

```
>>> student_dict = {'name':'Mimoy', 'Age':12,
'Grade':7, 'id':'7102'}
>>> student_dict['name']
'Mimoy'
```

Keys are case sensitive. If you give `student_dict['age']` instruction a key Error will be generated because the key actually has capital A in Age. The actual key is `Age` and not `age`.

```
>>> student_dict['age']
Traceback (most recent call last):
File "<pyshell#20>", line 1, in <module>
student_dict['age']
KeyError: 'age'
>>> student_dict['Age']
12
>>>
```

**Question: Can we change the value of an element of a dictionary?**

*Answer:* Yes, we can change the value of an element of a dictionary because dictionaries are mutable.

```
>>> dict1 = {'English
literature':'67%','Maths':'78%','Social
Science':'87%','Environmental Studies':'97%'}
>>> dict1['English literature'] = '78%'
>>> dict1
{'English literature': '78%', 'Maths': '78%',
'Social Science': '87%', 'Environmental Studies':
'97%'}
>>>
```

**Question: Can a dictionary have a list as a key or a value?**

*Answer:* A dictionary can have a list as value but not as a key.

```
>>> dict1 = {'English
literature':'67%','Maths':'78%','Social
Science':'87%','Environmental Studies':'97%'}
>>> dict1['English literature'] = ['67%', '78%']
>>> dict1
{'English literature': ['67%', '78%'], 'Maths':
'78%', 'Social Science': '87%', 'Environmental
Studies': '97%'}
>>>
```

If you try to have a list as a key then an error will be generated:

## 58 - Python Interview Questions

```
>>> dict1 = {[‘67%’, ‘78%’]:‘English literature’,  
‘Maths’: ‘78%’, ‘Social Science’: ‘87%’,  
‘Environmental Studies’: ‘97%’}  
Traceback (most recent call last):  
  File “<pyshell#30>”, line 1, in <module>  
dict1 = {[‘67%’, ‘78%’]:‘English literature’,  
‘Maths’: ‘78%’, ‘Social Science’: ‘87%’,  
‘Environmental Studies’: ‘97%’}  
TypeError: unhashable type: ‘list’  
>>>
```

### Question: How are elements deleted or removed from a dictionary?

Answer: Elements can be deleted or removed from a dictionary in any of the following ways:

```
>>> dict1 = {‘English literature’:‘67%’,‘Maths’:‘78%’,‘Social  
Science’:‘87%’,‘Environmental Studies’:‘97%’}
```

I. The `pop()` can be used to remove a particular value from a dictionary .  
`pop()` requires a valid key value as an argument or you will receive a key error. If you don't pass any argument then you will receive a type error:  
“`TypeError: descriptor ‘pop’ of ‘dict’ object needs an argument`”

```
>>> dict1.pop(‘Maths’)  
‘78%’  
>>> dict1  
{‘English literature’: ‘67%’, ‘Social Science’:  
‘87%’, ‘Environmental Studies’: ‘97%’}
```

II. The `popitem()` can be used to remove any arbitrary item.

```
>>> dict1.popitem()  
(‘Environmental Studies’, ‘97%’)  
>>> dict1  
{‘English literature’: ‘67%’, ‘Social Science’:  
‘87%’}
```

III. You can delete a particular value from a dictionary using ‘`del`’ keyword.

```
>>> del dict1[‘Social Science’]  
>>> dict1  
{‘English literature’: ‘67%’}
```

IV. `clear()` function can be used to clear the contents of a dictionary.

```
>>> dict1.clear()
>>> dict1
{}
```

V. The `del()` function can also be used to delete the whole dictionary after which if you try to access the dictionary object, a Name Error will be generated as shown in the following code:

```
>>> del dict1
>>> dict1
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>  dict1
NameError: name 'dict1' is not defined
>>>
```

**Question: What is `copy()` method used for?**

*Answer:* A copy method creates a shallow copy of a dictionary and it does not modify the original dictionary object in any way.

```
>>> dict2 = dict1.copy()
>>> dict2
{'English literature': '67%', 'Maths': '78%', 'Social Science': '87%', 'Environmental Studies': '97%'}
>>>
```

**Question: Explain from `Keys()` method.**

*Answer:* The `fromkeys()` method returns a new dictionary that will have the same keys as the dictionary object passed as the argument. If you provide a value then all keys will be set to that value or else all keys will be set to 'None'.

I. Providing no value

```
>>> dict2 = dict.fromkeys(dict1)
>>> dict2
{'English literature': None, 'Maths': None, 'Social Science': None, 'Environmental Studies': None}
>>>
```

II. Providing a value

```
>>> dict3 = dict.fromkeys(dict1, '90%')
```

```
>>> dict3
{'English literature': '90%', 'Maths': '90%',
'Social Science': '90%', 'Environmental Studies':
'90%'}
>>>
```

**Question: What is the purpose of *items()* function?**

*Answer:* The *items()* function does not take any parameters. It returns a view object that shows the given dictionary's key value pairs.

```
>>> dict1 = {'English
literature':'67%','Maths':'78%','Social
Science':'87%','Environmental Studies':'97%'}

>>> dict1.items()
dict_items([('English literature', '67%'),
('Maths', '78%'), ('Social Science', '87%'),
('Environmental Studies', '97%')])
>>>
```

**Question: *dict1 = {(1,2,3):['1','2','3']}***

**Is the instruction provided above, a valid command?**

*Answer:* *dict1 = {(1,2,3):['1','2','3']}* is a valid command. This instruction will create a dictionary object. In a dictionary the key must always have an immutable value. Since, the key is a tuple which is immutable, this instruction is valid.

**Question: *dict\_items([(‘English literature’, ‘67%’), (‘Maths’, ‘78%’), (‘Social Science’, ‘87%’), (‘Environmental Studies’, ‘97%’)])*.**

**Which function can be used to find total number of key-value pairs in *dict1*?**

*Answer:* The *len()* function can be used to find the total number of key-value pairs.

```
>>> dict1 = {'English
literature':'67%','Maths':'78%','Social
Science':'87%','Environmental Studies':'97%'}

>>> len(dict1)
4
>>>
```

**Question:** dict1 = {'English literature': '67%', 'Maths': '78%', 'Social Science': '87%', 'Environmental Studies': '97%'}

dict1.keys()

**What would be the output?**

**Answer:** The keys() function is used to display a list of keys present in the given dictionary object.

```
>>> dict1 = {'English literature': '67%', 'Maths': '78%', 'Social Science': '87%', 'Environmental Studies': '97%'}
>>> dict1.keys()
dict_keys(['English literature', 'Maths', 'Social Science', 'Environmental Studies'])
>>>
```

**Question:** dict1 = {'English literature': '67%', 'Maths': '78%', 'Social Science': '87%', 'Environmental Studies': '97%'}

**'Maths' in dict1**

**What would be the output?**

**Answer:** True

The 'in' operator is used to check if a particular key exists in the dict or not. If the key exists then it returns 'True' else it will return 'False'.

**Question:** dict1 = {'English literature': '67%', 'Maths': '78%', 'Social Science': '87%', 'Environmental Studies': '97%'}

dict2 = {(1,2,3):['1','2','3']}

dict3 = {'Maths': '78%'}

dict4 = {'Maths': '98%', 'Biology': '56%'}

**What would be the output of the following:**

1. dict1.update(dict2)
2. dict2 = {(1,2,3):['1','2','3']}
3. dict3.update(dict3)
4. dict1.update(dict4)

**Answer:** The update() method takes a dictionary object as an argument. If the keys already exists in the dictionary then the values are updated and if the key does not exist then the key value pair is added to the dictionary.

## 62 - Python Interview Questions

```
>>> dict1 = {'English literature':'67%','Maths':'78%','Social Science':'87%','Environmental Studies':'97%'}
>>> dict2 = {(1,2,3):['1','2','3']}
>>> dict1.update(dict2)
>>> dict1
{'English literature': '67%', 'Maths': '78%', 'Social Science': '87%', 'Environmental Studies': '97%', (1, 2, 3): ['1', '2', '3']}

dict1 = {'English literature':'67%','Maths':'78%','Social Science':'87%','Environmental Studies':'97%'}
>>> dict3 = {'Maths': '78%'}
>>> dict1.update(dict3)
>>> dict1
{'English literature': '67%', 'Maths': '78%', 'Social Science': '87%', 'Environmental Studies': '97%'}

dict3 = {'Maths': '78%'}
` 

>>> dict3.update(dict3)
>>> dict3
{'Maths': '78%'}
dict1 = {'English literature':'67%','Maths':'78%','Social Science':'87%','Environmental Studies':'97%'}
dict4 = {'Maths': '98%','Biology':'56%'}
dict1.update(dict4)
dict1
{'English literature': '67%', 'Maths': '98%', 'Social Science': '87%', 'Environmental Studies': '97%', 'Biology': '56%'}
```

**Question:** `dict1 = {'English literature':'67%','Maths':'78%','Social Science':'87%','Environmental Studies':'97%'}`

`dict1.values()`

What will be the output?

Answer:

```
>>> dict1.values()
dict_values(['67%', '78%', '87%', '97%'])
>>>
```

Sets

- Σ Unordered collection of items
    - Every element is unique and immutable
  - Σ Set itself is mutable
    - Used for performing set operations in Math
  - Σ Can be created by placing all the items in curly brackets {}, separated by ','
    - Σ Set can also be created using the inbuilt set() function
- ```
>>> set1 = {8,9,10}
>>> set1
{8, 9, 10}
>>>
```
- Σ Since sets are unordered, indexing does not work for it

**Question: How can we create an empty set?**

Answer: The inbuilt function set() is used to create an empty set. Empty curly braces cannot be used for this task as it will create an empty dictionary object.

```
>>> set1 = {}
>>> type(set1)
<class 'dict'>
>>> set1 = set()
>>> type (set1)
<class 'set'>
>>>
```

**Question: How can we add single element to a set?**

Answer: We can add single element with the help of add() function.

## 64 - Python Interview Questions

```
>>> set1 = {12, 34, 43, 2}
>>> set1.add(32)
>>> set1
{32, 2, 34, 43, 12}
>>>
```

### Question: How can we add multiple values to a set?

Answer: Multiple values can be added using update() function.

```
>>> set1 = {12, 34, 43, 2}
>>> set1.update([76, 84, 14, 56])
>>> set1
{2, 34, 43, 12, 76, 14, 84, 56}
>>>
```

### Question: What methods are used to remove value from sets?

Answer: (1) discard() (2) remove()

```
>>> set1 = {2, 34, 43, 12, 76, 14, 84, 56}
>>> set1.remove(2)
>>> set1
{34, 43, 12, 76, 14, 84, 56}
>>> set1.discard(84)
>>> set1
{34, 43, 12, 76, 14, 56}
>>>
```

### Question: What is the purpose of pop() method?

Answer: The pop() function randomly removes an element from a set.

```
>>> set1 = {2, 34, 43, 12, 76, 14, 84, 56}
>>> set1.pop()
2
>>> set1
{34, 43, 12, 76, 14, 84, 56}
>>>
```

### Question: How can we remove all elements from a set?

Answer: All elements from a set can be removed using clear() function.

```
>>> set1 = {2, 34, 43, 12, 76, 14, 84, 56}
>>> set1.clear()
>>> set1
set()
```

**Question: What are various set operations?**

*Answer:*

```
>>> #Union of two sets
>>> set1 = {1,5,4,3,6,7,10}
>>> set2 = {10,3,7,12,15}
>>> set1 | set2
{1, 3, 4, 5, 6, 7, 10, 12, 15}
>>> #Intersection of two sets
>>> set1 = {1,5,4,3,6,7,10}
>>> set2 = {10,3,7,12,15}
>>> set1 & set2
{10, 3, 7}
>>> #Set difference
>>> set1 = {1,5,4,3,6,7,10}
>>> set2 = {10,3,7,12,15}
>>> set1 - set2
{1, 4, 5, 6}
>>> #Set symmetric difference
>>> set1 = {1,5,4,3,6,7,10}
>>> set2 = {10,3,7,12,15}
>>> set1^set2
{1, 4, 5, 6, 12, 15}
>>>
```

# Chapter 3

## Operators in Python

What are operators?

Operators are required to perform various operations on data. They are special symbols that are required to carry out arithmetic and logical operations. The values on which the operator operates are called operands.

So, if we say  $10/5=2$

Here ‘/’ is the operator that performs division and 10 and 5 are the operands. Python has following operators defined for various operations:

1. Arithmetic Operators
2. Relational Operators
3. Logical/Boolean Operators
4. Assignment Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

**Question: What are Arithmetic Operators? What are various types of arithmetic operators that we can use in Python?**

**Answer:** Arithmetic operators are used to perform mathematical functions such as addition, subtraction, division, and multiplication. Various types of Arithmetic operators that we can use in Python are as follows:

‘+’ for Addition

```
>>> a = 9  
>>> b = 10  
>>> a + b  
19  
>>>
```

‘-’ for Subtraction

## 68 - Python Interview Questions

```
>>> a = 9  
>>> b = 10  
>>> a - b  
-1  
>>>
```

‘\*’ for Multiplication

```
>>> a = 9  
>>> b = 10  
>>> a * b  
90  
>>>
```

‘/’ for division

```
>>> a = 9  
>>> b = 10  
>>> a/b  
0.9  
>>>
```

‘%’ for Modulus – provides the value of remainder

```
>>> a = 9  
>>> b = 10  
>>> a % b  
9  
>>> a = 11  
>>> b = 2  
>>> a % b  
1  
>>>
```

‘//’ for Floor division – division of operands, provides integer value of quotient. The value after decimal points is removed.

```
>>> a = 9  
>>> b = 10  
>>> a // b  
0  
>>> a = 11  
>>> b = 2  
>>> a // b  
5  
>>>
```

‘\*\*’ for finding Exponential value

```
>>> a = 2  
>>> b = 3  
>>> a**b  
8  
>>> b**a  
9  
>>>
```

**Question: What is the Arithmetic operators precedence in Python?**

*Answer:* When more than one arithmetic operator appears in an expression the operations will execute in a specific order. In Python the operation precedence follows as per the acronym PEMDAS.

Parenthesis

Exponent

Multiplication

Division

Addition

Subtraction

$$(2+2)/2-2*2/(2/2)^2$$

$$= 4/2 - 4/1^2$$

$$= 2-8$$

$$= -6$$

$$>>> (2+2)/2-2*2/(2/2)^2$$

$$-6.0$$

**Question: a = 2, b =4, c = 5, d = 4 Evaluate the following keeping Python's precedence of operators:**

1. a+b+c+d
2. a+b\*c+d
3. a/b+c/d
4. a+b\*c+a/b+d

*Answer:*

```
>>> a=2
>>> b=4
>>> c=5
>>> d=4
>>> a+b+c+d
15
>>> a+b*c+d
26
>>> a/b+c/d 1.75
>>>
a+b*c+a/b+d
26.5
>>>
```

**Question: What are relational operators?**

**Answer:** Relational operators are also known as conditional or comparison operators. Relational operators in Python are defined as follows:

1. == : returns true if two operands are equal
2. !=: returns true if two operands are not equal
3. >: returns true if the left operand is greater than the right operand
4. <: returns true if the left operand is smaller than the right operand
5. >=: returns true if the left operand is greater than or equal to the right operand
6. <=: returns true if the left operand is smaller or equal to the right operand

```
>>> a = 5
>>> b = 6
>>> c = 7
>>> d = 7
>>> a == b
False
>>> c ==d
True
>>> a != b
True
>>> c != d
False
```

```
>>>  
>>> a > b  
False  
>>> a < b  
True  
>>> a >= b  
False  
>>> c >= d  
True  
>>> a <= b  
True  
>>> c <= d  
True
```

**Question:** a = 5, b = 6, c = 7, d=7

**What will be the outcome for the following:**

1. a<=b>=c
2. -a+b==c>d
3. b+c==6+d>=13

**Answer:**

```
>>> a <= b >= c  
False  
>>> -a + b == c > d  
False  
>>>  
b + c == 6 + d >= 13  
True
```

**Question:** What are assignment operators?

**Answer:** Assignment operators are used for assigning values to variables. Various types of assignment operators are as follows:

1. `=`: `a = 5` means that `a` is assigned value of 5
2. `+=` `a += 5` is same as `a = a + 5`
3. `-=`: `a -= 5` is same as `a = a - 5`
4. `*=`: `a *= 5` is same as `a = a * 5`
5. `/=`: `a /= 5` is same as `a = a / 5`
6. `%=`: `a %= 5` is same as `a = a % 5`

## 72 - Python Interview Questions

7.  $//=$ :  $x //= 5$  is same as  $x = x // 5$
8.  $**=$ :  $x **= 5$  is same as  $x = x ** 5$
9.  $&=$ :  $x &= 5$  is same as  $x = x & 5$
10.  $|=$ :  $x |= 5$  is same as  $x = x | 5$
11.  $^=$ :  $x ^= 5$  is same as  $x = x ^ 5$
12.  $>>=$ :  $x >>= 5$  is same as  $x = x >> 5$
13.  $<<=$ :  $x <<= 5$  is same as  $x = x << 5$

### **Question: Is $a = a * 2 + 6$ same as $a *= 2 + 6$ ?**

*Answer:* No,  $a = a * 2 + 6$  is not same as  $a *= 2 + 6$  this is because assignment operator have lower precedence than the addition operator. So, if  $a = 5$  then,

$$\begin{aligned}a &= a * 2 + 6 \Rightarrow a = 16 \\a &*= 2 + 6 \Rightarrow a = 40\end{aligned}$$

```
>>> a = 5
>>> a = a *2+6
>>> a
16
>>> a = 5
>>> a *= 2 + 6
>>> a
40
>>>
```

### **Question: What are logical operators?**

*Answer:* Logical operators are generally used in control statements like if and while. They are used to control program flow. Logical operator evaluate a condition and return “True” or “False” depending on whether the condition evaluates to True or False. Three logical operators in Python are as follows:

- ‘and’
- ‘or’ and
- ‘not’

```
>>> a = True  
>>> b = False  
>>> a and b  
False  
>>> a or b  
True  
>>> not a  
False  
>>> not b  
True  
>>>
```

### Question: What are membership operators?

Answer: The membership operators are used to check if a value exists in a sequence or not.

Two types of membership operators are as follows:

1. in: returns true if a value is found in a sequence
2. not in: returns true if a value is not found in a sequence

```
>>> a = "Hello World"  
>>> "h" in a  
False  
>>> "H" in a  
True  
>>> "h" not in a  
True  
>>> "H" not in a  
False  
>>>
```

### Question: What are bitwise operators?

Answer: Bitwise operators work on bits and perform bit by bit operations. In Python, the following bit-wise operations are defined:

1. AND - &  
 $2 \& 3$   
 $2$
2. OR - |  
 $2|3$   
 $3$
3. One's complement - ~

## 74 - Python Interview Questions

```
>>> ~2  
-3  
  
4. XOR - ^  
2^3  
1  
5. Right shift - >>  
2>>2  
0  
6. Left shift - <<  
2<<2  
8
```

### Question: What are identity operators?

Answer: Identity operators are used to verify whether two values are on the same part of the memory or not. There are two types of identity operators:

1. is: return true if two operands are identical
2. is not: returns true if two operands are not identical

```
>>> a = 3  
>>> id(a)  
140721094570896  
>>> b = 3  
>>> id(b)  
140721094570896  
>>> a is b  
True  
>>> a = 3  
>>> b = 6  
>>> c = b - a  
>>> id(c)  
140721094570896  
>>> a is c  
True  
>>> a = 4  
>>> b = 8  
>>> a is b  
False  
>>> a is not b  
True  
>>>
```

**Question: What is the difference between `a = 10` and `a==10`?**

*Answer:* The expression `a = 10` assigns the value 10 to variable `a`, whereas `a == 10` checks if value of `a` is equal to 10 or not. If yes then it returns ‘True’ else it will return ‘False’.

**Question: What is an expression?**

*Answer:* Logical line of code that we write while programming, are called expressions. An expression can be broken into operator and operands. It is therefore said that an expression is a combination of one or more operands and zero or more operators that are together used to compute a value.

For example:

`a = 6`

`a + b = 9`

`8/7`

**Question: What are the basic rules of operator precedence in Python?**

*Answer:* The basic rule of operator precedence in Python is as follows:

1. Expressions must be evaluated from left to right.
2. Expressions of parenthesis are performed first.
3. In Python the operation precedence follows as per the acronym PEMDAS:
  - a. Parenthesis
  - b. Exponent
  - c. Multiplication
  - d. Division
  - e. Addition
  - f. Subtraction
4. Mathematical operators are of higher precedence and the Boolean operators are of lower precedence. Hence, mathematical operations are performed before Boolean operations.

**Question: Arrange the following operators from high to low precedence:**

1. Assignment
2. Exponent
3. Addition and Subtraction
4. Relational operators

5. Equality operators
6. Logical operators
7. Multiplication, division, floor division and modulus

*Answer:* The precedence of operators from high to low is as follows:

1. Exponent
2. Multiplication, division, floor division and modulus
3. Addition and subtraction operators
4. Relational operators
5. Equality operators
6. Assignment operators
7. Logical Operators

**Question: Is it possible to change the order of evaluation in an expression?**

*Answer:* Yes, it is possible to change order of evaluation of an expression. Suppose you want to perform addition before multiplication in an expression, then you can simply put the addition expression in parenthesis.

$$(2+4)*4$$

**Question: What is the difference between implicit expression and explicit expression?**

*Answer:* Conversion is the process of converting one data type into another. Two types of conversion in Python are as follows:

1. Implicit type conversion
2. Explicit type conversion

When Python automatically converts one data type to another it is called implicit conversion.

```
>>> a = 7
>>> type(a)
<class 'int'>
>>> b = 8.7
>>> type(b)
<class 'float'>
>>> type(a+b)
<class 'float'>
>>>
```

Explicit conversion is when the developer has to explicitly convert datatype of an object to carry out an operation.

```
>>> c = "12"
>>> type(c)
<class 'str'>
>>> d = 12
# addition of string and integer will generate error

>>> c+d
Traceback (most recent call last):
File "<pyshell#43>", line 1, in <module>
c+d
TypeError: can only concatenate str (not "int") to str
# convert string to integer and then add

>>> int(c)+d
24
# convert integer to string and then perform
concatenation
>>> c+str(d)
'1212'
>>>
```

#### **Question: What is a statement?**

*Answer:* A complete unit of code that Python interpreter can execute is called a statement.

#### **Question: What is input statement?**

*Answer:* The input statement is used to get user input from the keyboard. The syntax for input() function is as follows:

`input(prompt)`

The prompt is a string message for the user.

```
>>> a = input("Please enter your message here :")
Please enter your message here : It is a beautiful
day
>>> a
' It is a beautiful day'
>>>
```

## 78 - Python Interview Questions

Whenever an input function is called, the program comes on hold till an input is provided by the user. The input() function converts the user input to a string and then returns it to the calling program.

### **Question: Look at the following code:**

```
num1 = input("Enter the first number: ")
num2 = input("Enter the second number: ")
print(num1 + num2)
```

When the code is executed the user provides the following values:

Enter the first number: 67

Enter the second number: 78

What would be the output?

*Answer:* The output will be 6778. This is because the input() function converts the user input into a string and then returns it to the calling program. So, even though the users has entered integer values, the input() function has returned string values '67' and '78' and the '+' operator concatenates the two strings giving '6778' as answer. To add the two numbers they must be first converted to a integer value. Hence, the code requires slight modification:

```
num1 = input("Enter the first number: ")
num2 = input("Enter the second number: ")
print(int(num1) + int(num2))
```

Output:

```
Enter the first number: 67
Enter the second number: 78
145
>>>
```

### **Question: What is Associativity of Python Operators? What are non-associative operators?**

*Answer:* Associativity defines the order in which an expression will be evaluated if it has more than one operators having same precedence. In such a case generally left to right associativity is followed.

Operators like assignment or comparison operators have no associativity and are known as Non associative operators.

## Chapter 4

# Decision Making and Loops

### Control Statements

Control statements are used to control the flow of program execution. They help in deciding the next steps under specific conditions also allow repetitions of program for certain number of times.

Two types of control statements are as follows:

#### 1. Conditional branching

Σ If

Syntax:

```
if (condition): to do  
statement
```

If the condition evaluates to be true only then if code under if block will be executed.

Σ if...else

Syntax:

```
if (condition):  
to do statement  
else:  
to do statement  
Σ Nested if statements
```

Syntax:

```
if (condition1):  
    to do statement  
elif(condition2):  
    else do this  
elif(condition3):  
    else do this
```

#### 2. Loops

Σ while: repeat a block of statements as long as a given condition is true

Syntax:

```
while(condition):
    to do statement
Σ for: repeat a block of statements for certain number of times
```

Syntax:

```
for < iterating_variable > in sequence:
Repeat these steps
    Σ nested loops
```

**Question: What would be the output for the following piece of code?**

```
animals = ['cat', 'dog'] for
pet in animals:
    pet.upper()
print(animals)
```

*Answer:* The output will be ['cat', 'dog']. The value returned by pet.upper() is not assigned to anything hence it does not update the value in any way.

**Question: What would be the output of the following code?**

```
for i in range(len(animals)):
    animals[i] = animals[i].upper()
print(animals)
```

*Answer:* ['CAT', 'DOG']

**Question: What would be the output of the following code?**

```
numbers = [1,2,3,4]
for i in numbers:
    numbers.append(i + 1)
print(numbers)
```

*Answer:* This piece of code will not generate any output as the 'for' loop will never stop executing. In every iteration, one element is added to the end of the list and the list keeps growing in size.

**Question: What will be the output for the following code?**

```
i = 6
while True:
    if i%4 == 0:
        break
    print(i)
    i -= 2
```

Answer: 6

**Question:** Write a code to print the following pattern:

```
*  
**  
***  
****
```

Answer:

```
for i in range(1,5):  
    print("****i")
```

Or

```
count = 1  
while count < 5:  
    print('****count)  
    count = count + 1
```

**Question:** Write code to produce the following pattern:

```
1  
22  
333  
4444
```

Answer: The code will be as follows:

```
count = 1  
while count < 5:  
    print(str(count)*count)  
    count = count + 1
```

**Question:** Write a code to generate the following pattern:

```
1  
12  
123  
1234
```

Answer: The code will be as follows:

```
count = 1
string1 =""
while count < 5:
    for i in range(1, count+1):
        string1 = string1+str(i)    count =
        count + 1
    print(string1)
    string1 =""
```

**Question:** Write code to spell a word entered by the user.

*Answer:* The code will be as follows:

```
word = input("Please enter a word : ")
for i in word:
    print(i)
```

Output

Please enter a word : Aeroplane

A  
e  
r  
o  
p  
l  
a  
n  
e

**Question:** Write code to reverse a string.

*Answer:* The code:

```
string1 = "AeRoPIAnE"
temp = list(string1)
count = len(temp)-1
reverse_str=""
while count>=0:
    reverse_str = reverse_str + temp[count]
    count = count-1
print(reverse_str)
```

## Output

```
EnAlPoReA
```

### Statements to control a loop

The following three statements can be used to control a loop:

1. break: breaks the execution of the loop and jumps to next statement after the loop
2. continue: takes the control back to the top of the loop without executing the remaining statements
3. pass: does nothing

### Question: What will be the output for the following code?

```
a = 0
for i in range(5): a =
    a+1
    continue
    print(a)
```

Answer: 5

### Question: What would be the output for the following code?

Answer: The code:

```
for item in ('a','b','c','d'):
    print (item)
    if item == 'c':
        break
    continue
    print ("challenge to reach here")
```

### Question: How would you use a "if " statement to check whether an integer is even ?

Answer:

Code

```
x = int(input("enter number : "))
if x%2 == 0:
    print("You have entered an even number")
```

## 84 - Python Interview Questions

Output

```
enter number : 6
You have entered an even
number >>>
```

**Question: How would you use a “if ” statement to check whether an integer is odd?**

*Answer:*

Code

```
x = int(input("enter number : "))
if x%2 != 0:
    print("You have entered an odd number")
```

Output

```
enter number : 11
You have entered an odd number
```

**Question: Use if-else statement to check if a given number is even if yes then display that a message stating that the given number is even else print that the given number is odd.**

*Answer:* Please have a look at the following code:

Code

```
x = int(input("enter number : "))
if x%2 == 0:
    print("You have entered an even number")
else:
    print("You have entered an odd number")
```

Output

```
enter number : 11
You have entered an odd
number >>>

enter number : 4
You have entered an even
number >>>
```

**Question: What is a ternary operator?**

*Answer:* Ternary operator is a conditional expression used to compress the if ...else block in one single line.

```
[to do if true] if [Expression] else [to do if false]
```

Code

```
x = 27
print("You have entered an even number") if x%2
== 0 else print("You have entered an odd number")
```

Output

```
You have entered an odd number
```

**Question: What would be the output of the following code? Why?**

```
i = j = 10
if i > j:
    print("i is greater than j")
elif i<= j:
    print("i is smaller than j")
else:
    print("both i and j are equal")
```

*Answer:* The output of the above code will be:

i is smaller than j

i is equal to j.

So, the second condition elif i>j evaluates to true and so, the message printed in this block is displayed.

**Question: How can the following piece of code be expressed in one single line?**

```
i = j = 10
if i > j:
    print("i is greater than j")
elif i< j:
    print("i is smaller than j")
else:
    print("both i and j are equal")
```

*Answer:* print ("i is greater than j" if i > j else "i is smaller than j" if i < j else"both i and j are equal")

**Question: What will be the output for the following code?**

```
i = 2  
j = 16  
minimum_val = i < j and i or j  
minimum_val
```

*Answer:* 2

**Question: What is the meaning of conditional branching?**

*Answer:* Deciding whether certain sets of instructions must be executed or not based on the value of an expression is called conditional branching.

**Question: What would be the output for the following code?**

```
a = 0  
b = 9  
i = [True, False][a > b]  
print(i)
```

*Answer:* The answer would be “True”. This is another ternary syntax:

```
[value_if_false, value_if_true][test_condition]
```

In the above code  $a < b$ , therefore the test\_condition is false. Hence, ‘i’ will be assigned the value of value\_if\_false which in this case is set to “True”.

**Question: What is the difference between the *continue* and *pass* statement?**

*Answer:* pass does nothing whereas continue starts the next iteration of the loop.

# Chapter 5

## User Defined Functions

While going through the chapter on standard data types you have learnt about several inbuilt defined by functions. These functions already exist in Python libraries. However, programming is all about creating your own functions that can be called any time. A function is basically a block of code that will execute only when it is called. To define a function we use the keyword def as shown in the following code:

```
def function_name ():  
    to do statements
```

So, let's define a simple function:

```
def new_year_greetings():  
    print("Wish you a very happy and properous new  
year")  
new_year_greetings()
```

The new\_year\_greetings() function is a very simple function that simply displays a new year message when called.

You can also pass a parameter to a function. So, if you want the function new\_year\_greetings() to print a personalized message, we may want to consider passing a name as a parameter to the function.

```
def new_year_greetings(name):  
    print("Hi ",name.upper(),"!! Wish you a very happy  
and properous new year")  
    name = input("Hello!! May I know your good name  
please : ")  
    new_year_greetings(name)
```

The output for the code given above would be as follows:

```
Hello!! May I know your good name please : Jazz  
Hi JAZZ !! Wish you a very happy and properous new  
year  
>>>
```

## 88 - Python Interview Questions

Indentation is very important. In order to explain, let's add another print statement to the code.

```
def new_year_greetings(name):
    print("Hi ",name.upper(),"!! Wish you a very happy
and properous new year")
    print("Have a great year")
    name = input("Hello!! May I know your good name
please : ")
    new_year_greetings(name)
```

So, when the function is called, the output will be as follows:

```
Hello!! May I know your good name please : Jazz
Hi JAZZ !! Wish you a very happy and properous
new year
Have a great year
```

Improper indentation can change the meaning of the function:

```
def new_year_greetings(name):
    print("Hi ",name.upper(),"!! Wish you a very happy
and properous new year")
    print("Have a great year")
```

In the code given above the second print statement will be executed even if the function is not called because, it is not indented properly and it is no more part of the function. The output will be as follows:

```
Have a great year
```

Multiple functions can also be defined and one function can call the other.

```
def new_year_greetings(name):
    print("Hi ",name.upper(),"!! Wish you a very happy
and properous new year")
    extended_greetings()
def extended_greetings():
    print("Have a great year ahead")
    name = input("Hello!! May I know your good name
please : ")
    new_year_greetings(name)
```

When `new_year_greeting()` a message is printed and then it calls `extended_greetings()` function which prints another message.

Output

```
Hello!! May I know your good name please : Jazz
```

Hi JAZZ !! Wish you a very happy and properous new year

Have a great year ahead

Multiple parameters can also be passed to a function.

```
def new_year_greetings(name1,name2):
    print("Hi ",name1," and ",name2,"!! Wish you a very
    happy and properous new year")
    extended_greetings()
def extended_greetings():
    print("Have a great year ahead")
new_year_greetings("Jazz","George")
```

The output will be as follows:

Hi Jazz and George !! Wish you a very happy and properous new year

Have a great year ahead

**Question: What are the different types of functions in Python?**

*Answer:* There are two types of functions in Python:

1. Built-in functions: library functions in Python
2. User-defined functions: defined by the developer

**Question: Why are functions required?**

*Answer:* Many times in a program a certain set of instructions may be called again and again. Instead of writing the same piece of code where it is required it is better to define a function and place the code in it. This function can be called whenever there is a need. This saves time and effort and the program can be developed easily. Functions help in organizing coding work and testing of code also becomes easy.

**Question: What is a function header?**

*Answer:* First line of function definition that starts with `def` and ends with colon (`:`) is called a function header.

**Question: When does a function execute?**

*Answer:* A function executes when a call is made to it. It can be called directly from the Python prompt or from another function.

**Question: What is a parameter? What is the difference between a parameter and argument?**

**Answer:** A parameter is a variable that is defined in a function definition whereas an argument is an actual value that is passed on to the function. The data carried in the argument is passed on to the parameters. An argument can be passed on as a literal or as a name.

```
def function_name(param):
```

In the preceding statement, param is a parameter. Now, take a look at the statement given below, it shows how a function is called:

```
function_name(arg):
```

arg is the data that we pass on while calling a function. In this statement arg is an argument.

So, a parameter is simply a variable in method definition and an argument is the data passed on the method's parameter when a function is called.

#### **Question: What is a default parameter?**

**Answer:** Default parameter is also known as optional parameter. While defining a function if a parameter has a default value provided to it then it is called a default parameter. If while calling a function the user does not provide any value for this parameter then the function will consider the default value assigned to it in the function definition.

#### **Question: What are the types of function arguments in Python?**

**Answer:** There are three types of function arguments in Python:

1. Default Arguments: assumes a default value, if no value is provided by the user.

```
def func(name = "Angel"):
    print("Happy Birthday ", name)
func()
Happy Birthday Angel
```

You can see that the default value for name is “Angel” and since the user has not provided any argument for it, it uses the default value.

2. Keyword Arguments: We can call a function and pass on values irrespective of their positions provided we use the name of the parameter and assign them values while calling the function.

```
def func(name1, name2):
    print("Happy Birthday", name1, " and "
          ",name2,!!!")
```

Output:

```
func(name2 = "Richard",name1 = "Marlin")
Happy Birthday Marlin and Richard !!!
```

3. Variable-length Arguments: If there is an uncertainty about how many arguments might be required for processing a function we can make use of variable-length arguments. In the function definition, if a single '\*' is placed before parameter then all positional arguments from this point to the end are taken as a tuple. On the other hand if "\*\*" is placed before the parameter name then all positional arguments from that point to the end are collected as a dictionary.

```
def func(*name, **age):
    print(name)
    print(age)
func("Lucy","Aron","Alex", Lucy = "10",Aron =
"15",Alex="12")
```

Output:

```
('Lucy', 'Aron', 'Alex')
{'Lucy': '10', 'Aron': '15', 'Alex': '12'}
```

#### **Question: What is a fruitful and non-fruitful function?**

**Answer:** Fruitful function is a function that returns a value and a non-fruitful function does not return a value. Non-fruitful functions are also known as void function.

#### **Question: Write a function to find factorial of a number using for loop**

**Answer:** The code for finding a factorial using for loop will be as follows:

Code

```
def factorial(number):
    j = 1
    if number==0|number==1:
        print(j)
    else:
        for i in range (1, number+1):
            print(j," * ",i," = ",j*i) j = j*i
        print(j)
```

## 92 - Python Interview Questions

### Execution

```
factorial(5)
```

### Output

```
1 * 1 = 1
1 * 2 = 2
2 * 3 = 6
6 * 4 = 24
24 * 5 = 120
120
```

### Question: Write a function for Fibonacci series using a for loop:

Answer: Fibonacci series: 0, 1, 1, 2, 3, 5, 8....

We take three variables:

i, j, and k:

- If i = 0, j = 0, k = 0
- If i = 1, j = 1, k = 0
- If i > 1:
  - temp = j
  - j = j + k
  - k = temp

The calculations are as shown as follows:

| i | k | j                                                   |
|---|---|-----------------------------------------------------|
| 0 | 0 | 0                                                   |
| 1 | 0 | 1                                                   |
| 2 | 0 | temp = j = 1<br>j = j + k = 1+0 = 1<br>k = temp = 1 |
| 3 | 1 | temp = j = 1<br>j = j + k = 1+1 = 2<br>k = temp = 1 |
| 4 | 1 | temp = j = 2<br>j = j + k = 2+1 = 3<br>k = temp = 2 |

|   |   |                                                  |
|---|---|--------------------------------------------------|
| 5 | 2 | temp = j =3<br>j = j + k = 3+2 =5 k<br>= temp =3 |
| 6 | 3 | temp = j =5<br>j = j + k = 5+3 =8<br>k = temp =1 |

Code

```
def fibonacci_seq(num):  
    i = 0  
    j = 0  
    k = 0  
    for i in range(num): if  
i==0:  
    print(j)  
  
    elif i==1:  
    j = 1  
    print(j)  
  
    else:  
    temp = j  
    j = j+k  
    k = temp  
    print(j)
```

Execution

```
fibonacci_seq(10)
```

Output

```
0  
1  
1  
2  
3  
5  
8  
13  
21  
34
```

**Question: How would you write the following code using while loop?**

[Note: You may want to refer to recursion before attempting this question.]

```
def test_function(i,j):
    if i == 0:
        return j;
    else:
        return test_function(i-1,j+1)
print(test_function(6,7))
```

*Answer:*

```
def test_function(i,j):
    while i > 0:
        i = i- 1
        j = j+1
    return j
print(test_function(6,7))
```

**Question: Write code for finding the HCF of two given numbers.**

*Answer:* HCF stands for *Highest Common Factor* or *Greatest Common Divisor* for two numbers. This means that it is the largest number within the range of 1 to smaller of the two given numbers that divides the two numbers perfectly giving the remainder as zero.

1. Define a function hcf() that takes two numbers as input.

```
def hcf(x,y):
```

2. Find out which of the two numbers is greatest, the other one will be the smallest.

```
small_num = 0
if x > y:
    small_num = y else:
    small_num = x
```

Set a for loop for the range 1 to small\_num+1. (We take the upper limit as small\_num+1 because the for loop operates for one number less than the upper limit of the range). In this for loop, divide both the numbers with each number in the range and if any number divides both, perfectly assign that value to hcf as shown in the following code:

```
for i in range(1,small_num+1):  
    if (x % i == 0) and (y % i == 0):  
        hcf = i
```

Suppose, the two numbers are 6 and 24, first both numbers are divisible by 2. So, hcf = 2, then both numbers will be divisible by 3 so, the value of 3 will be assigned to 3. Then the loop will encounter 6, which will again divide both the numbers equally. So, 6 will be assigned to hcf. Since, the upper limit of the range has reached, the function will finally have hcf value of 6.

3. Return the value of hcf:

```
return hcf
```

Code

```
def hcf(x,y):  
    small_num = 0  
    if x > y:  
        small_num = y else:  
        small_num = x  
  
    for i in range(1,small_num+1):  
        if (x % i == 0) and (y % i == 0): hcf = i  
    return hcf
```

Execution

```
print(hcf(6,24))
```

Output

```
6
```

Scope of a variable

Scope of a variable can be used to know which program can be used from which section of a code. The scope of a variable can be local or global.

Local variables are defined inside a function and global functions are defined outside a function. Local variables can be accessed only within

## 96 - Python Interview Questions

the function in which they are defined. Global variable can be accessed throughout the program by all functions.

```
total = 0 # Global variable
def add(a,b):
    sumtotal = a+b #Local variable
    print("inside total = ",total)
```

### Question: What will be the output of following code?

```
total = 0
def add(a,b):
    global total
    total = a+b
    print("inside total = ",total)

add(6,7)
print("outside total = ",total)
```

Answer: The output will be as follows:

```
inside total = 13
outside total = 13
```

### Question: What would be the output for the following code?

```
total = 0
def add(a,b):
    total = a+b
    print("inside total = ",total)

add(6,7)
print("outside total = ",total)
```

Answer:

```
inside total = 13
outside total = 0
```

### Question: Write the code to find HCF using Euclidean Algorithm.

Answer:

The following figure shows two ways to find HCF.

On the left hand side you can see the traditional way of finding the HCF.

On the right hand side is the implementation of Euclidean Algorithm to find HCF.

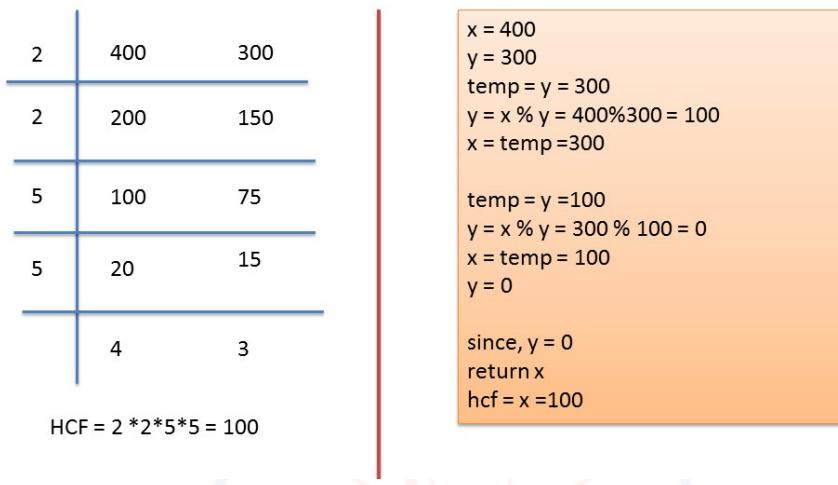


Figure 15

Code

```
def hcf(x,y):
    small_num = 0
    greater_num = 0
    temp = 0
    if x > y:
        small_num = y
        greater_num = x
    else:
        small_num = x
        greater_num = y

    while small_num > 0:
        temp = small_num
        small_num = greater_num % small_num
        greater_num = temp
    return temp
```

## 98 - Python Interview Questions

### Execution

```
print("HCF of 6 and 24 = ",hcf(6,24))
print("HCF of 400 and 300 = ",hcf(400,300))
```

### Output

HCF of 6 and 24 = 6

HCF of 400 and 300 = 100

### Question: Write code to find all possible palindromic partitions in a string.

Answer: The code to find all possible palindromic partitions in a string will involve the following steps:

1. Create a list of all possible substrings.
2. Substrings are created by slicing the strings as all possible levels using for loop.
3. Every substring is then checked to see if it is a palindrome.
4. The substring is converted to a list of single characters.
5. In reverse order the characters from the list are added to a string.
6. If the resultant string matches the original string then it is a palindrome.

### Code

```
def create_substrings(x):
    substrings = []

    for i in range(len(x)):
        for j in range(i, len(x)+1):
            if x[i:j] != "":
                substrings.append(x[i:j])
    for i in substrings:
        check_palin(i)

def check_palin(x):
    palin_str = ""
    palin_list = list(x)
    y = len(x)-1
    while y>=0:
        palin_str = palin_str + palin_list[y]
        y = y-1
    if(palin_str == x):
        print("String ", x, " is a palindrome")
```

## Execution

```
x = "malayalam"  
create_substrings(x)
```

## Output

```
String m is a palindrome  
String malayalam is a palindrome  
String a is a palindrome  
String ala is a palindrome String  
alayala is a palindrome String l is a  
palindrome  
String layal is a palindrome String a  
is a palindrome  
String aya is a palindrome String y  
is a palindrome  
String a is a palindrome  
String ala is a palindrome String l is  
a palindrome  
String a is a palindrome  
String m is a palindrome
```

### Question: What are anonymous functions?

Answer: Lambda facility in Python can be used for creating function that have no names. Such functions are also known as anonymous functions. Lambda functions are very small functions that have just one line in function body. It requires no return statement.

```
total = lambda a, b: a + b  
total(10,50)  
60
```

### Question: What is the use of return statement?

Answer: The return statement exits function and hands back value to the function's caller. You can see in the code given below. The function func() returns sum of two numbers. This value assigned to "total" and then the value of total is printed.

```
def func(a,b):  
    return a+b  
    total = func(5,9)  
    print(total)
```

14

**Question: What will be the output of the following function?**

```
def happyBirthday():
    print("Happy Birthday")

a = happyBirthday()
print(a)
```

*Answer:*

Happy Birthday  
None

**Question: What will be the output of the following code?**

```
def outerWishes():
    global wishes
    wishes = "Happy New Year"
    def innerWishes():
        global wishes
        wishes = "Have a great year ahead"
        print('wishes =', wishes)
    wishes = "Happiness and Prosperity Always"
    outerWishes()
    print('wishes =', wishes)
```

*Answer:* The output will be as follows:

wishes = Happy New Year

**Question: What is the difference between passing immutable and mutable objects as argument to a function?**

*Answer:* If immutable arguments like strings, integers, or tuples are passed to a function, the object reference is passed but the value of these parameters cannot be changed. It acts like pass by value call. Mutable objects too are passed by object reference but their values can be changed.

# Chapter 6

## Classes and Inheritance

### Modules

- Σ Modules are used to create a group of functions that can be used by anyone on various projects.
- Σ Any file that has python code in it can be thought of as a Module.
- Σ Whenever we have to use a module we have import it in the code.
- Σ Syntax:

```
import module_name
```

### Object Orientation

- Σ Object Orientation programming helps in maintaining the concept of reusability of code.
- Σ Object Oriented Programming languages are required to create a readable and reusable code for complex programs.

### Classes

- Σ A class is a blueprint for the object.
- Σ To create a class we use the Keyword *class*.
- Σ The class definition is followed by the function definitions.

```
class class_name:
```

```
def function_name(self):
```

### Components of a class

A class would consist of the following components:

1. class Keyword
2. instance and class attributes
3. self keyword
4. `_init_` function

### Instance and class attribute

Class attributes remain same for all objects of the class whereas instance variables are parameters of `__init__()` method. These values are different for different objects.

#### The self

- Σ It is similar to this in Java or pointers in C++.
- Σ All functions in Python have one extra first parameter (the ‘self’) in function definition, even when any function is invoked no value is passed for this parameter.
- Σ If there a function that takes no arguments, we will have to still mention one parameter – the “self” in the function definition.

#### The `__init__()` method:

- Σ Similar to constructor in Java
- Σ `__init__()` is called as soon as an object is instantiated. It is used to initialize an object.

#### Question: What will be the output of the following code?

```
class BirthdayWishes:  
    def __init__(self, name):  
        self.name = name  
    def bday_wishes(self):  
        print("Happy Birthday ",self.name,"!!")  
    bdaywishes = BirthdayWishes("Christopher")  
    bdaywishes.bday_wishes()
```

Answer: The output will be as follows:

```
Happy Birthday Christopher !!
```

#### Question: What are class variables and instance variables?

Answer: Class and instance variables are defined as follows:

```
class Class_name:  
    class_variable_name = static_value  
  
        def __init__(instance_variable_val):  
            Instance_variable_name = instance_  
            variable_val
```

Class variables have the following features:

- They are defined within class construction
- They are owned by class itself
- They are shared by all instances in class
- Generally have same value for every instance
- They are defined right under the class header
- Class variables can be accessed using dot operator along with the class name as shown below:

Class\_name.class\_variable\_name

The instance variables on the other hand:

- Are owned by instances.
- Different instances will have different values for instance variables.
- To access the instance variable it is important to create an instance of the class:

instance\_name = Class\_name()

instance\_name.Instance\_variable\_name

**Question: What would be the output of the following code?**

```
class sum_total:
    def calculation(self, number1,number2 = 8):
        return number1 + number2
st = sum_total()
print(st.calculation(10,2))
```

**Answer:**

12

**Question: When is \_\_init\_\_0 function called ?**

**Answer:** The \_\_init\_\_0 function is called when the new object is instantiated.

**Question: What will be the output of the following code?**

```
class Point:
    def __init__(self, x=0,y=10,z=0): self.x =
        x + 2
        self.y = y + 6
        self.z = z + 4
p = Point(10, 20,30)
print(p.x, p.y, p.z)
```

Answer:

12 26 34

**Question: What will be the output for the following code?**

```
class StudentData:  
    def __init__(self, name, score, subject):  
        self.name = name  
        self.score = score  
        self.subject = subject  
    def getData(self):  
        print("the result is {0}, {1}, {2}."  
              format(self.name, self.score, self.subject))  
sd = StudentData("Alice", 90, "Maths")  
sd.getData()
```

Answer:

the result is Alice, 90, Maths

**Question: What will be the output for the following code?**

```
class Number_Value:  
    def __init__(self, num):  
        self.num = num  
    num = 500  
    num = Number_Value(78.6)  
    print(num.num)
```

Answer:

78.6

### Inheritance

Object Oriented languages allow us to reuse the code. Inheritance is one such way that takes code reusability to another level all together. In inheritance we have a superclass and a subclass. The subclass will have attributes that are not present in superclass. So, imagine that we are making a software program for a dog Kennel. For this we can have a dog class that has features that are common in all dogs. However, when we move on to specific breeds there will be differences in each breed. So, we can now create classes for each breed. These classes will inherit common features of the dog class and to those features, it will add its own attributes

that makes one breed different from the other. Now, let's try something. Let's go step by step. Create a class and then create its subclass to see how things work. Let's use a simple example so that it is easy for you to understand the mechanism behind it.

Step 1:

Let's first define a class using "Class" Keyword as shown below:

```
class dog():
```

Step 2:

Now that a class has been created, we can create a method for it. For this example we create one simple method which when invoked prints a simple message *I belong to a family of Dogs*.

```
def family(self):  
    print("I belong to family of Dogs")
```

The code so far looks like the following:

```
class dog():  
    def family(self):  
        print("I belong to family of Dogs")
```

Step 3:

In this step we create an object of the class dog as shown in the following code:

```
c = dog()
```

Step 4:

The object of the class can be used to invoke the method family() using dot '.' operator as shown in the following code:

```
c.family()
```

At the end of step 4 the code would look like the following:

```
class dog():  
    def family(self):  
        print("I belong to family of Dogs")  
  
c = dog()  
c.family()
```

## 106 - Python Interview Questions

When we execute the program we get the following output:

I belong to family of Dogs

From here we move on to implementation of the concept of inheritance. It is widely used in object oriented programming. By using the concept of inheritance you can create a new class without making any modification to the existing class. The existing class is called the base and the new class that inherits it will be called derived class. The features of the base class will be accessible to the derived class.

We can now create a class germanShepherd that inherits the class dog as shown in the following code:

```
class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")
```

The object of class germanShepherd can be used to invoke methods of class dog as shown in the following code:

```
Final program
class dog():
    def family(self):
        print("I belong to family of Dogs")
class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")

c = germanShepherd()
c.family()
c.breed()
```

Output

```
I belong to family of Dogs
I am a German Shepherd
```

If you look at the code above, you can see that object of class germanShepherd can be used to invoke the method of class.

Here are few things that you need to know about inheritance.

Any number of classes can be derived from a class using inheritance.

In the following code we create another derived class Husky. Both the classes germanShepherd and husky call the family method of dog class and breed method of their own class.

```
class dog():
    def family(self):
        print("I belong to family of Dogs")

class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")

class husky(dog):
    def breed(self):
        print("I am a husky")
    g = germanShepherd()
    g.family()
    g.breed()
    h = husky()
    h.family()
    h.breed()
```

Output

```
I belong to family of Dogs
I am a German Shepherd
I belong to family of Dogs
I am a husky
```

A derived class can override any method of its base class.

```
class dog():
    def family(self):
        print("I belong to family of Dogs")

class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")
    class husky(dog):
        def breed(self):
```

```
print("I am a husky")
def family(self):
    print("I am class apart")

g = germanShepherd()
g.family()
g.breed()
h = husky()
h.family()
h.breed()
```

Output

```
I belong to family of Dogs I
am a German Shepherd
I am class apart
I am a husky
```

A method can call a method of the base class with same name.

Look at the following code, the class husky has a method family() which call the family() method of the base class and adds its own code after that.

```
class dog():
    def family(self):
        print("I belong to family of Dogs")

class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")

class husky(dog):
    def breed(self):
        print("I am a husky")
    def family(self):
        super().family()
        print("but I am class apart")

g = germanShepherd()
g.family()
g.breed()
h = husky()
h.family()
h.breed()
```

## Output

```
I belong to family of Dogs  
I am a German Shepherd  
I belong to family of Dogs  
but I am class apart  
I am a husky
```

### **Question: What is multiple inheritance?**

*Answer:* If a class is derived from more than one class, it is called multiple inheritance.

### **Question: A is a subclass of B. How can one invoke the \_\_init\_\_ function in B from A?**

*Answer:* The \_\_init\_\_ function in B can be invoked from A by any of the two methods:

- super().\_\_init\_\_()
- \_\_init\_\_(self)

### **Question: How in Python can you define a relationship between a bird and a parrot.**

*Answer:* Inheritance. Parrot is a subclass of bird.

### **Question: What would be the relationship between a train and a window?**

*Answer:* Composition

### **Question: What is the relationship between a student and a subject?**

*Answer:* Association

### **Question: What would be the relationship between a school and a teacher?**

*Answer:* Composition

### **Question: What will be the output for the following code:**

```
class Twice_multiply:  
    def __init__(self):  
        self.calculate(500)  
  
    def calculate(self, num):  
        self.num = 2 * num;  
class Thrice_multiply(Twice_multiply):  
    def __init__(self):  
        super().__init__()  
        print("num from Thrice_multiply is", self.num)  
  
    def calculate(self, num):  
        self.num = 3 * num;  
tm = Thrice_multiply()
```

Answer:

```
num from Thrice_multiply is 1500  
>>>
```

**Question: For the following code is there any method to verify whether tm is an object of Thrice\_multiply class?**

```
class Twice_multiply:  
    def __init__(self):  
        self.calculate(500)  
    def calculate(self, num):  
        self.num = 2 * num;  
class Thrice_multiply(Twice_multiply):  
    def __init__(self):  
        super().__init__()  
        print("num from Thrice_multiply is", self.num)  
    def calculate(self, num):  
        self.num = 3 * num;  
tm = Thrice_multiply()
```

**Answer** Yes, one can check whether an instance belongs to a class or not using `isinstance()`

```
isinstance(tm, Thrice_multiply)
```

# Chapter 7

## Files

Till now you have learnt how to implement logic in Python to write blocks of code that can help you accomplish certain tasks. In this section we will learn about how to use Python to [work with files](#). You can read data from files and you can also write data to the files. You can not only access the internet but also check your emails and social media accounts using Python programming language.

### Files

A file has a permanent location. It exists somewhere on the computer disk and can be referred to anytime. It is stored on the hard disk in non-volatile memory which means the information in the file remains even if the computer is switched off. In this section we will learn how to deal with files in Python.

### Open a File

If you want to work on an existing file, then you will have to first open the file. In this section we will learn how to open a file.

In order to open a file we will make use of `open()` function which is an inbuilt function. When we use `open()` function, a file object is returned which can be used to read or modify the file. If the file exists in the same directory where python is installed then you need not give the entire path name. However, if the location is different then you will have to mention the entire path.

For this example I have created a file by the name: `learning_files.txt` in the current directory of python `/home/pi`.

The content of the file is as follows:

I am great a learning files

See how Good I am at opening Files

Thank you Python

Look at the following piece of code:

```
>>> f_handle = open("learning_files.txt")
>>> print(f_handle.read())
```

In case the file is not available, the error message will be displayed as follows:

```
>>> f_handle = open("llllearning_files.txt")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    f_handle = open("llllearning_files.txt")
FileNotFoundError: [Errno 2] No such file or
directory: ' llllearning_files.txt'
>>>
```

Output

```
I am great a learning files
See how Good I am at opening Files
Thank you Python
```

### Python File Modes

Python has defined file modes which can be implemented when a file is opened. These mode define what can be done with the file once it is opened. If you do not mention the mode then “read” is considered the default mode. List of various modes is given as follows:

‘r’ is also the default mode, it means that the file has been opened for reading purpose. You have already seen the usage of read mode. To explain this a file by the name “test.txt” has been created. The content of the file are as follows:

*“I am excited about writing on the file using Python for the first time. Hope You feel the same.”*

We will now give the following commands.:.

```
>>> f_handle = open("test.txt",'r') >>>
f_handle.read(4)
```

The output for the above code is :

‘I am’

`f_handle.read(4)` retrieves first four characters from the file and displays it. ‘w’ stands for writing. It means that you want to open a file and write in it. In case the file that you have mentioned does not exist then a new file will be created.

```
>>> f_handle = open("test.txt",'w')
>>> f_handle.write("I am excited about writing on
the file using Python for the first time.")
71
>>> f_handle.write("Hope you feel the same.")
22
>>> f_handle.close()
>>>
```

So, if you open the file now this is how the contents would look like:

The original content of the file before passing the write instructions was:  
“*I am excited about writing on the file using Python for the first time. Hope you feel the same.*”  
If you open the file after passing “write” instructions now the contents of the file will be as follows:  
“*Hi I have opened this file again and I feel great again.*”

As you can see that the previous lines (*I am excited about writing on the file using Python for the first time. Hope you feel the same.*) have been erased from the file.

Now, we close the file and try to write something again in it.

```
>>> f_handle = open(test.txt",'w')
>>> f_handle.write("\n Hi I have opened this file
again and I feel great again.")
58
>>> f_handle.close()
>>>
```

‘x’ stands for exclusive creation. An error will be displayed if the file already exists. Let’s see what happens when we try to use ‘x’ mode with already existing test.txt file.

```
>>> f_handle = open("F:/test.txt",'x')
```

```
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    f_handle = open("F:/test.txt",'x')
  FileExistsError: [Errno 17] File exists: 'F:/test.
txt'
```

'a' is used to append an existing file. If a file does not exist then a new file will be created.

So, now we try to add new text to an already existing file.

The content of test.txt file is as follows:

*"I am excited about writing on the file using Python for the first time.  
Hope You feel the same."*

We will try to add the following line to it:

*"Hi I have opened this file again and I feel great again."*

In order to append, we follow the following steps:

```
>>> f_handle = open("test.txt",'a')
>>> f_handle.write("Hi I have opened this file
again and I feel great again.")
56
>>> f_handle.close()
>>>
```

Output

I am excited about writing on the file using Python for the first time.  
Hope You feel the same.

Hi I have opened this file again and I feel great again.

't' is used to open a file in text mode and 'b' is used to open the file in binary mode.

In the above examples you must have noticed f\_handle.close() command.

It is important to use the close() command after you have stopped working with a file in order to free up operating system resources. If you leave the file open, you may encounter problems.

A better way of dealing with files is to keep the code related to file reading in a try block as shown in the following code:

```
>>> try:
    f_handle = open("IIIlearning_files.txt")
    content = f_handle.read()
    f_handle.close()
except IOError:
    print("Could not find the file. Please check
again")
exit()
```

Output

**Could not find the file. Please check again**

In the above piece of code the file name given, does not exist in the given location. Hence, the remaining code of the try block is ignored and the code written in the except block is applied. In the except block we have provided a simpler and user friendly message which is easier for the user to understand. Without the try expect block the following message will be displayed:

```
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
f_handle = open("IIIlearning_files.txt")
FileNotFoundException: [Errno 2] No such file or
directory: 'IIIlearning_files.txt'
```

Which can be difficult to decipher.

File-system-type commands:

Now we will have a look at some very common file-system-type operations such as move, copy, and so on.

We now try to copy the contents of this file to another file. For this we require to import shutil as shown in the following code:

```
>>> import shutil
>>>         shutil.copy("F:/test.txt","F:/test1.txt")
'F:/test1.txt'
```

Output:

Content of test1.txt file:

You can move the file or change the name of the file using move command as shown in the following code:

```
>>> import shutil  
>>>     shutil.move("test.txt","test2.txt")  
'test2.txt'  
>>>
```

The above set of instructions changes the name of the file test.txt to test2.txt.

Another important package available with Python is glob.

The glob package allows you to create a list of particular type of files by using star \* operator.

```
>>> import glob  
>>> glob.glob("*txt")  
['important.txt', 'learning_files.txt', 'test1.txt',  
'test2.txt']  
>>>
```

## Chapter

# Algorithm Analysis and Big-O

### Algorithm

An algorithm is a procedure or set of instructions that are carried out in order to solve a problem. Coding demands procedural knowledge which is expressed in the form of algorithms. An algorithm provides a recipe or a roadmap or a simple sequence of instructions that are required to accomplish a particular programming task. So, for things as simple as adding two numbers or as complex as designing a banking program, everything requires a systematic approach. It is important to create a roadmap which makes sense before going ahead with the coding part. For simple programs it is easy to create algorithm by logical thinking. There are also some famous algorithms that can be used for complex problem solving and hence frequently used for coding.

#### **Question: What are the steps involved in writing an algorithm?**

*Answer:* There are three main steps involved in writing an algorithm:

##### 1. Data Input

- Formulate the problem and decide what data types will be the input.

##### 2. Data Processing

- Decide what computation will be carried out to get the desired result.

- Also identify the decision points, conditions under which various functions will operate.

##### 3. Results Output

- One should know the expected results so that it can be verified by the actual results.

#### **Question: What are the characteristics of Algorithm?**

*Answer:* An algorithm has following five characteristics:

1. Precision: It clearly defines one single starting point and one or more well defined ending points.
2. Effectiveness: It is composed of effective primitives that are understood by the person or machine using it.
3. Input/Output Specified: The algorithm must receive an input and it must also produce an output.
4. Finiteness: An algorithm stops after execution of finite number of steps.
5. Uniqueness: In an algorithm the result of every step is uniquely defined and its value depends only on the input provided and the output generated by the previous steps.

**Question: What is the meaning of Problem solving?**

*Answer:* Problem solving is a logical process in which a problem is first broken down into smaller parts that can be solved step by step to get the desired solution.

**Question: What would you call an algorithm that puts element lists in certain order?**

*Answer:* An algorithm that puts elements of a list in certain order is called sorting algorithm. It uncovers a structure by sorting the input. Sorting is often required for optimizing the efficiency of other algorithms. The output of a sorting algorithm is in non decreasing order where no element is smaller than the original element of the input and the output reorders but retains all the elements of the input which is generally in array form.

Some very commonly known sorting algorithms are as follows:

1. Simple sorts:
  - a. Insertion sort
  - b. Selection sort
2. Efficient Sorts:
  - a. Merge sort
  - b. Heap sort
  - c. Quick sort
  - d. Shell sort
3. Bubble sort
4. Distribution sort
- a. Counting sort

- b. Bucket sort
- c. Radix sort

**Question: What type of algorithm calls itself with smaller or simpler input values?**

*Answer:* A recursive algorithm calls itself with smaller input values. It is used if a problem can be solved by using its own smaller versions.

**Question: What is the purpose of divide and conquer algorithm? Where is it used?**

*Answer:* As the name suggests the divide and conquer algorithm divides the problem into smaller parts. These smaller parts are solved and the solutions obtained are used to solve the original problem. It is used in Binary search, Merge sort, quick sort to name a few.

**Question: What is dynamic programming?**

*Answer:* In a dynamic problem, an optimization problem is broken down into much simpler sub-problems, each sub problem is solved only once and the result is stored. For example, Fibonacci Sequence (Explained in Chapter on Recursion).

### Big-O Notation

Big-O notation helps you analyse how an algorithm would perform if the data involved in the process increases or in simple words it is simplified analysis of how efficient an algorithm can be.

Since, algorithms are an essential part of software programming it is important for us to have some idea about how long an algorithm would take to run, only then can we compare two algorithms and a good developer would always consider time complexity while planning the coding process.

It helps in identifying how long an algorithm takes to run.

### Big – O

- (1) Gives us the algorithm complexity in terms of input size n.
- (2) It considers only the steps involved in the algorithm.
- (3) Big-O analysis can be used to analyse both time and space.

An algorithm's efficiency can be measured in terms of best- average or worst case but Big-O notation goes with the worst case scenario.

## 120 - Python Interview Questions

It is possible to perform a task in different ways which means that there can be different algorithms for the same task having different complexity and scalability.

Now, suppose there are two functions:

### Constant Complexity [O(1)]

A task that is constant will never experience variation during runtime, irrespective of the input value. For Example:

```
>>> x = 5 + 7  
>>> x  
12  
>>>
```

The statement  $x = 5+7$  does not depend on the input size of data in any way. This is known as O(1)(big oh of 1).

Example:

Suppose, there are sequence of steps all of constant time as shown in the following code:

```
>>> a=(4-6)+ 9  
>>> b = (5 * 8) +8  
>>> print(a * b)  
336  
>>>
```

Now let's compute the Big-O for these steps:

Total time =  $O(1)+O(1)+O(1)$

=  $3O(1)$

While computing Big – O we ignore constants because once the data size increases, the value of constant does not matter.

Hence the Big-O would be O(1).

### Linear complexity: [O(n)]

In case of linear complexity the time taken depends on the value of the input provided.

Suppose you want to print table of 5. Have a look at the following code:

```
>>> for i in range(0,n):  
    print("\n 5 x ",i,"=",5*i)
```

The number of lines to be printed, depends on 'n'. For n =10, only ten lines will be printed but for n= 1000, the for loop will take more time to execute.

The print statement is O(1).

So, the block of code is  $n \cdot O(1)$  which is O(n).

Consider following lines of code:

```
>>>j = 0 ----- (1)
>>> for i in range(0,n): ----- (2)
```

`print("\n 5 x ",i,"=",5*i)`

(1) is O(1)

(2) block is O(n)

Total time = O(1)+O(n)

We can drop O(1) because it is a lower order term and as the value of n becomes large (1) will not matter and the runtime actually depends on the for loop.

Hence the Big-O for the code mentioned above is O(n).

Quadratic Complexity:[O( $n^2$  )]

As the name indicates quadratic complexity, the time taken depends on the square of the input value. This can be the case with nested loops. Look at the following code:

```
>>> for i in range (0,n):
    for j in range(0,n):
        print("I am in ", j," loop of i = ", i,".")
```

In this piece of code the print statement is executed  $n^2$  times.

Logarithmic Complexity

Logarithmic complexity indicates that in worst case the algorithm will have to perform  $\log(n)$  steps. To understand this, let's first understand the concept of logarithm.

Logarithms are nothing but inverse of exponentiation.

Now let's take a term  $2^3$ . Here 2 is the base and 3 is the exponent. We therefore say that base 2 log of 8 ( $\log_2 8$ ) = 3. Same way if then base 10 log of 100000 then base 10 log of 100000 ( $\log_{10} 100000$ ) = 5.

Since the computer works with binary numbers, therefore in programming and Big O we generally work with base 2 logs.

Have a look at the following observation:

$$1. 2^0 = 1, \log_2 1 = 0$$

$$2. 2^1 = 2, \log_2 2 = 1,$$

3.  $2^2 = 4$ ,  $\log_2 4 = 2$ ,

4.  $2^3 = 8$ ,  $\log_2 8 = 3$ ,

5.  $2^4 = 16$ ,  $\log_2 16 = 4$ ,

This means that if  $n = 1$ , number of steps = 1.

If  $n = 4$ , number of steps will be 2.

If  $n = 8$ , then number of steps will be 3.

So, as data doubles the number of steps increase by one.

The number of steps grow slowly in comparison to the growth in data size.

The best example for logarithmic complexity in software programming is Binary Search tree. You will learn more about it in chapter based on Trees.

**Question: What is worst-case time complexity of an algorithm?**

**Answer:** The worst-case time complexity in computer science means worst-case in terms of time consumed while execution of a program. It is the longest running time that an algorithm can take. The efficiency of algorithms can be compared by looking at the order of growth of the worst-case complexity.

**Question: What is the importance of Big-O notation?**

**Answer:** Big-O notation describes how fast the runtime of an algorithm will increase with respect to the size of the input. It would give you a fair idea about how your algorithm would scale and also give you an idea about the worst-case complexity of the algorithms that you might be considering for your project. You would be able to compare two algorithms and decide which would be a better option for you.

**Question: What is the need for run time analysis for an algorithm when you can find the exact runtime for a piece of code?**

**Answer:** Exact runtime is not considered because the results can vary depending on the hardware of the machine, speed of the processor and the other processors that are running in the background. What is more important to understand is that how the performance of the algorithm gets affected with increase in input data.

**Question: Big-O analysis is also known as \_\_\_\_\_.**

**Answer:** Asymptotic analysis

**Question: What do you understand by asymptotic notation?**

*Answer:* It is very important to know how fast a program would run and as mentioned earlier we do not want to consider the exact run time because different computers have different hardware capabilities and we may not get the correct information in this manner.

In order to resolve this issue, the concept of asymptotic notation was developed. It provides a universal approach for measuring speed and efficiency of an algorithm. For applications dealing with large inputs we are more interested in behaviour of an algorithm as the input size increases. Big O notation is one of the most important asymptotic notations.

**Question: Why is Big O notation expressed as O(n)?**

*Answer:* Big-O notation compares the runtime for various types of input sizes. The focus is only on the impact of input size on the runtime which is why we use the  $n$  notation. As the value of  $n$  increases, our only concern is to see how it affects the runtime. If we had been measuring the runtime directly then the unit of measurement would have been time units like second, micro-second, and so on. However, in this case ' $n$ ' represents the size of the input and 'O' stands for 'Order'. Hence,  $O(1)$  stands for order of 1,  $O(n)$  stands for order of  $n$  and  $O(n^2)$  stands for order of the square of the size of input.

Big-O notations and names:

1. Constant –  $O(1)$
2. Logarithmic –  $O(\log(n))$
3. Linear –  $O(n)$
4. Log Linear –  $O(n\log(n))$
5. Quadratic –  $O(n^2)$
6. Cubic –  $O(n^3)$
7. Exponential –  $O(2^n)$

**Question: Write a code in python that takes a list of alphabets such as ['a', 'b', 'c', 'd', 'e', 'f', 'g'] and returns a list of combination such as ['bcdefg', 'acdefg', 'abdefg', 'abcefg', 'abcdfg', 'abcdeg', 'abcdef']. Find the time complexity for the code.**

*Answer:*

Code

```
input_value = input("enter the list of alphabets  
separate by comma : ")  
alphabets = input_value.split(',')  
final = []  
str = ""  
for element in range(0,len(alphabets)):  
    for index, item in alphabets:  
        if(item != alphabets[element]):  
            str = str+item  
    final.append(str)  
    str=""
```

Execution

```
print(final)
```

Output

```
enter the list of alphabets seperate by comma : a,b,c,d,e,f,g  
['bcdefg', 'acdefg', 'abdefg', 'abcefg', 'abcdgf', 'abcdeg', 'abcdef']  
>>>
```

Conclusion:

The code has a runtime of  $O(n^2)$ .

**Question: The following code finds the sum of all elements in a list. What is its time complexity?**

```
def sum(input_list):  
    total = 0  
    for i in input_list:  
        total = total + i  
    print(total)
```

*Answer:*

```
def sum(input_list):
```

```
    total = 0
```

-----(1)

```
    for i in input_list:
```

-----

(2)

```
total = total + i  
print(total) -----(3)  
time for block (1) = O(1)  
time for block (2) = O(n)  
time for block (3) = O(1)  
Total time = O(1) + O(n) + O(1)  
Dropping constants  
Total time = O(n)
```

**Question: What are the pros and cons of time complexity?**

*Answer:*

Pros:

It is a good way of getting an idea about the efficiency of the algorithm.

Cons:

It can be difficult to assess complexity for complicated functions.

**Question: What is the difference between time and space complexity?**

*Answer:* Time complexity gives an idea about the number of steps that would be involved in order to solve a problem. The general order of complexity in ascending order is as follows:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)$$

Unlike time, memory can be reused and people are more interested in how fast the calculations can be done. This is one main reason why time complexity is often discussed more than space complexity. However, space complexity is never ignored. Space complexity decides how much memory will be required to run a program completely. Memory is required for:

1. Instruction space
2. Knowledge space for constants, variables, structured variables, dynamically altered areas etc
3. Execution

**Question: What is the difference between auxillary space and space complexity?**

*Answer:* Auxillary space is the extra space that is required while executing an algorithm and it is temporary space.

Space complexity on the other hand is the sum of auxillary space and input space:

$$\text{Space complexity} = \text{Auxiliary space} + \text{Input space}$$

**Question: What is memory usage while execution of algorithm?**

**Answer:** Memory is used for:

1. Saving the compiled version of instructions.
2. In case of nested functions or one algorithm, calling another algorithm, the variables are pushed to a system stack and made to wait till the internal algorithm has finished executing.
3. To store data and variables.

### Space Complexity

You have learnt about time complexity. Now, let's move on to space complexity. As the name suggests space complexity describes how much memory space would be required if size of input n increases. Here too we consider the worst case scenario.

Now have a look at the following code:

```
>>> x = 23                                     (1)
>>> y = 34                                     (2)
>>> sum = x + y                                (3)
>>> sum
57
>>>
```

In this example we need space to store three variables: x in (1), y in (2), and sum in (3). However, this scenario will not change, and the requirement for 3 variables is constant and hence the space complexity is O(1).

Now, have a look at the following code:

```
word = input("enter a word : ")
word_list = []
for element in word:
    print(element)
    word_list.append(element)
print(word_list)
```

The size of the word\_list object increase with n. Hence, in this case the space complexity is O(n).

If there is a function 1 which has suppose three variables and this function1 calls another function named function2 that has 6 variables then the overall requirement for temporary workspace is of 9 units. Even if function1 calls function2 ten times the workspace requirement will remain the same.

**Question: What would be the space complexity for the following piece of code? Explain.**

```
n = int(input("provide a number : "))
statement = "Happy birthday"
for i in range(0,n):
    print(i+1,".",statement)
```

*Answer:* The space complexity for the above code will be O(1) because the space requirement is only for storing value of integer variable n and string statement. Even if the size of n increases the space requirement remains constant. Hence, O(1).

**Question: What is the time and space complexity for the following:**

```
a = b = 0
for i in range(n): a
= a + i
for j in range(m):
    b = b + j
```

*Answer:*

Timecomplexity

Time for first loop = O(n)

Time for second loop = O(m)

Total time = O(n) + O(m) = O(n + m)

Space complexity

O(1)

**Question: Find the time complexity for the following code:**

```
a = 0
for i in range(n):
    a = a + i
for j in range(m): a = a
    + i + j
```

*Answer:* Time complexity:  $O(n^2)$

**Question: What will be the time complexity for the following code?**

```
i = j = k = 0
for i in range(n/2,n):
    for j in range(2,n):
        k = k+n/2
        j = j*2
```

*Answer:*

Time complexity for the first for loop  $O(n/2)$ .

Time complexity for second for loop:  $O(\log n)$  because  $j$  is doubling itself till it is less than  $n$ .

$$\begin{aligned}\text{Total time} &= O(n/2) * O(\log n) \\ &= O(n \log n)\end{aligned}$$

**Question: What is the time complexity for the following code:**

```
i = a = 0
while i>0:
    a = a+i
    i = i/2
```

*Answer:* Time complexity will be  $O(\log n)$ .

Big O for Python Data Structures

Python language comprises of mutable and immutable objects. Numbers, strings, and tuple come in the latter category whereas lists, sets, and dictionary data type are mutable objects. The reason why lists and dictionaries are called mutable is because they can be easily altered anytime. They are like array because the data elements can be inserted or deleted easily by providing an index value and since the values can be easily added or removed from any point, the lists are considered to be dynamic. Hence, lists are known to be dynamic arrays. You can also say that lists are called dynamic array because:

1. Its size can be dynamically modified at run time.
2. You need not define the size of the list when you create it.
3. They allow you to store more than one variable.
4. Dynamic arrays, once filled allocate bigger chunk of memory and elements of the original array are copied to this section of memory as a result it can continue to fill available slots.

The most important operations that are performed on a list are as follows:

1. Indexing
2. Assigning value to an index value

Both the methods mentioned above are designed to run at constant time  $O(1)$ . The Big-O for common list operations are given as follows:

1. `index[]` :  $O(1)$
2. `index assignment` :  $O(1)$
3. `append`:  $O(1)$
4. `pop()` :  $O(1)$
5. `pop(i)`:  $O(n)$
6. `insert(i, item)`:  $O(n)$
7. `delete operator`:  $O(n)$
8. `contains(in)`:  $O(n)$
9. `get slice[x:y]`:  $O(k)$
10. `del slice`:  $O(n)$
11. `set slice` :  $O(n+k)$
12. `reverse`:  $O(n)$
13. `concatenate`:  $O(k)$
14. `sort`:  $O(n \log n)$
15. `multiply`  $O(nk)$

## Dictionaries

Dictionaries are implementation of hashtables and can be operated with keys and values.

Big-O efficiencies for common dictionary objects are given as follows:

1. `copy` :  $O(n)$
2. `get item` :  $O(1)$
3. `set item` :  $O(1)$
4. `delete item` :  $O(1)$
5. `contains(in)` :  $O(1)$
6. `iteration`:  $O(n)$

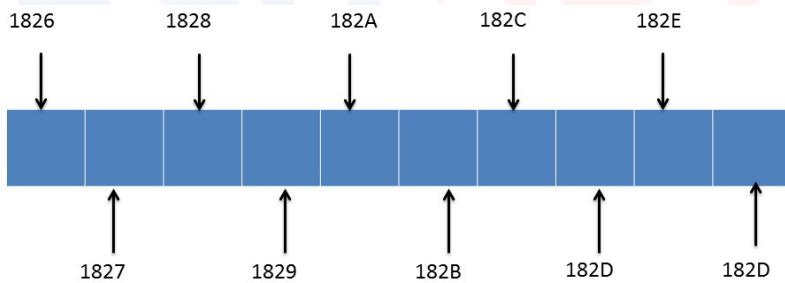
# Chapter 8

## Array Sequence

### Low Level Arrays

Let's try to understand how information is stored in low-level computer architecture in order to understand how array sequence work. Here is a small brush up on computer science basics on memory units.

We know that the smallest unit of data in a computer is a bit (0 or 1). 8 bits together make a byte. A byte has 8 binary digits. Characters such as alphabets, numbers, or symbols are stored in bytes. A computer system memory has huge number of bytes and tracking of how information is stored in these bytes is done with the help of memory address. Every byte has a unique memory address which makes tracking of information easier. The following diagram depicts a lower level computer memory. It shows a small section of memory of individual bytes with consecutive addresses.



**Illustration of Memory Addressing**

*Figure 16*

Computer system hardware is designed in such a manner that the main memory can easily access any byte in the system. The primary memory is

located in the CPU itself and is known as RAM. Any byte irrespective of the address can be accessed easily. Every byte in memory can be stored or retrieved in constant time, hence its Big-O notation for the time complexity would be O(1).

There is a link between an identifier of a value and the memory address where it is stored, the programming language keeps a track of this association. So, a variable `student_name` may store name details for a student and `class_teacher` would store name of a class teacher. While programming, it is often required to keep a track of all related objects. So, if you want to keep a track of score in various subjects for a student, then it is a wise idea to group these values under one single name, assign each value an index and use the index to retrieve the desired value. This can be done with the help of Arrays. An array is nothing but a contiguous block of memory.

Python internally stores every unicode character in 2 bytes. So, if you want to store 5-letter word (let's say 'state') in python, this is how it would get stored in memory:

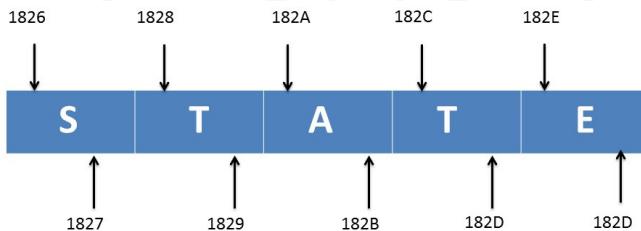


Figure 17

Since each Unicode character occupies 2 bytes, the word `STATE` is stored in 10 consecutive bytes in the memory. So, this is a case of array of 5 characters. Every location of an array is referred to as a cell. Every array element is numbered and its position is called index. In other words index describes the location of an element.

| Index | Element | Memory Location |
|-------|---------|-----------------|
| 0     | S       | 1826 & 1827     |
| 1     | T       | 1828 & 1829     |
| 2     | A       | 182A & 182B     |
| 3     | T       | 182C & 182D     |
| 4     | E       | 182E & 182D     |

Every cell of an array must utilize the same number of bytes.

The actual address of the 1st element of the array is called Base Address. Let's say the name of the array mentioned above is `my_array[]`. The Base address of `my_array[]` is 1826. If we have this information it is easy to calculate address of any element in the array.

Address of `my_array[index]` = *Base Address + (Storage size in bytes of one element in the array) \* index*.

Therefore, Address of `my_array[3]` =  $1826 + 2 \times 3$

$$= 1826 + 6$$

$$= 182C$$

After that, slight information on how things work at lower level let's get back to the higher level of programming where the programmer is only concerned with the elements and index of the array.

## Referential Array

We know that in an array, every cell must occupy same number of bytes. Suppose we have to save string values for food menu. The names can be of different length. In this case we can try to save enough space considering the longest possible name that we can think of but that does not seem to be a wise thing to do as lot of space is wasted in the process and you never know there may be a name longer than the value that we have catered for. A smarter solution in this case would be to use an array of object references.

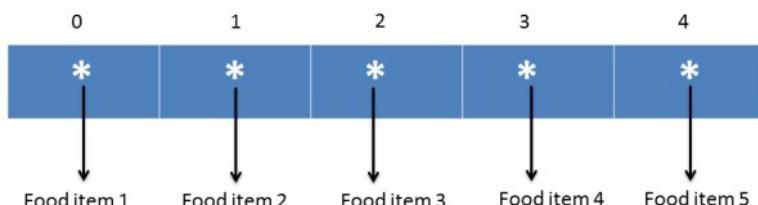


Figure 18

In this case every element of the array is actually reference to an object. The benefit of this is that every object which is of string value can be of different length but the addresses will occupy same number of cells. This helps in maintaining constant time factor of order O(1).

In Python, lists are referential in nature. They store pointers to addresses in the memory. Every memory address requires a space of 64-bits which is fixed.

**Question:** You have a list `integer_list` having integer values. What happens when you give the following command?

```
integer_list[1] += 7
```

*Answer:* In this case the value of the integer at index 1 does not change rather we end up referring to space in the memory that stores the new value i.e. sum of `integer_list[1]+7`.

**Question:** State whether True or False:

A single list instance may include multiple references to the same object as elements of the list.

*Answer:* True

**Question:** Can a single object be an element of two or more lists?

*Answer:* Yes

**Question:** What happens when you compute a slice of list?

*Answer:* When you compute slice of list, a new list instance is created. This new list actually contains references to the same elements that were in the parent list.

For example:

```
>>> my_list = [1, 2, 8, 9, "cat", "bat", 18] >>>
slice_list = my_list[2:6]
>>> slice_list
[8, 9, 'cat', 'bat']
>>>
```

This is shown in the following diagram:

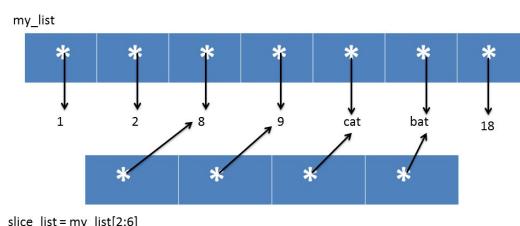


Figure 19

**Question:** Suppose we change the value of element at index 1 of `slice_list` to 18 (preceding diagram). How will you represent this in a diagram?

*Answer:* When we say `slice_list[1]=18`, we actually are changing the reference that earlier pointed to 9 to another reference that points to value 18. The actual integer object is not changed, only the reference is shifted from one location to the other.

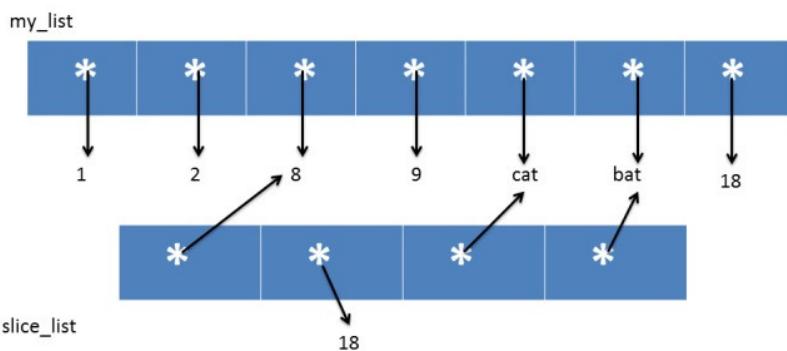


Figure 20

### Deep Copy and Shallow Copy in Python

Python has a module named “copy” that allows to deep copy or shallow copy mutable objects.

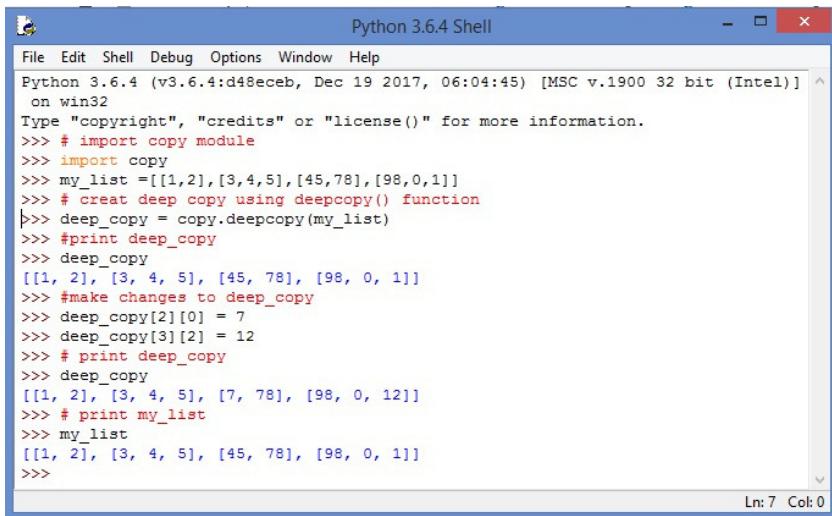
Assignment statements can be used to create binding between a target and an object, however they cannot be used for copying purposes.

#### Deep Copy

The `copy` module of python defines a `deepcopy()` function which allows the object to be copied into another object. Any changes made to the new object will not be reflected in the original object.

In case of shallow copy, a reference of the object is copied into another object as a result of which changes made in the copy will be reflected in the parent copy. This is shown in the following code:

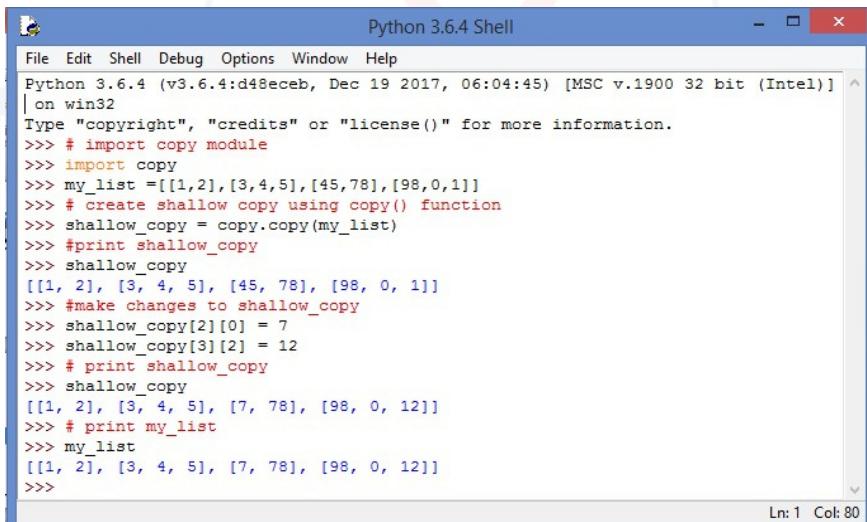
## 136 - Python Interview Questions



The screenshot shows the Python 3.6.4 Shell window. The code demonstrates creating a deep copy of a list and modifying its elements. The output shows that changes to the copied list do not affect the original list.

```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
| on win32
Type "copyright", "credits" or "license()" for more information.
>>> # import copy module
>>> import copy
>>> my_list =[[1,2],[3,4,5],[45,78],[98,0,1]]
>>> # creat deep copy using deepcopy() function
>>> deep_copy = copy.deepcopy(my_list)
>>> #print deep_copy
>>> deep_copy
[[1, 2], [3, 4, 5], [45, 78], [98, 0, 1]]
>>> #make changes to deep_copy
>>> deep_copy[2][0] = 7
>>> deep_copy[3][2] = 12
>>> # print deep_copy
>>> deep_copy
[[1, 2], [3, 4, 5], [7, 78], [98, 0, 12]]
>>> # print my_list
>>> my_list
[[1, 2], [3, 4, 5], [45, 78], [98, 0, 1]]
>>>
```

Figure 21



The screenshot shows the Python 3.6.4 Shell window. The code demonstrates creating a shallow copy of a list and modifying its elements. The output shows that changes to the copied list affect the original list.

```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
| on win32
Type "copyright", "credits" or "license()" for more information.
>>> # import copy module
>>> import copy
>>> my_list =[[1,2],[3,4,5],[45,78],[98,0,1]]
>>> # create shallow copy using copy() function
>>> shallow_copy = copy.copy(my_list)
>>> #print shallow_copy
>>> shallow_copy
[[1, 2], [3, 4, 5], [45, 78], [98, 0, 1]]
>>> #make changes to shallow_copy
>>> shallow_copy[2][0] = 7
>>> shallow_copy[3][2] = 12
>>> # print shallow_copy
>>> shallow_copy
[[1, 2], [3, 4, 5], [7, 78], [98, 0, 12]]
>>> # print my_list
>>> my_list
[[1, 2], [3, 4, 5], [7, 78], [98, 0, 12]]
>>>
```

Figure 22

It is important to note here that shallow and deep copying functions should be used when dealing with objects that contain other objects (lists or class instances). A shallow copy will create a compound object and insert into

it, the references the way they exist in the original object. A deep copy on the other hand creates a new compound and recursively inserts copies of the objects the way they exist in the original list.

**Question: What would be the result of following statement?**

`my_list = [7]*10`

*Answer:* It would create a list named `my_list` as follows:

`[7, 7, 7, 7, 7, 7, 7, 7, 7, 7]`

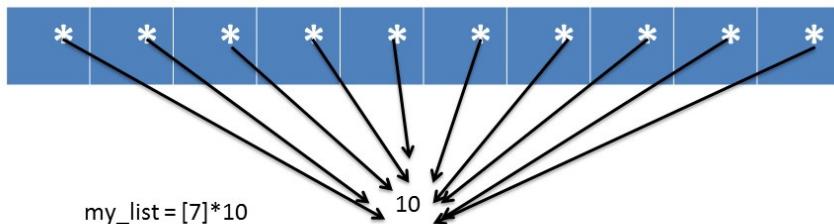


Figure 23

All the 10 cells of the list `my_list`, refers to the same element which in this case is 10.

**Question: Have a look at the following piece of code:**

```
>>> a = [1,2,3,4,5,6,7] >>>
b = a
>>> b[0] = 8
>>> b
[8, 2, 3, 4, 5, 6, 7] >>> a
[8, 2, 3, 4, 5, 6, 7] >>>
```

Here, we have used assignment operator still on making changes to 'b' changes are reflected in 'a'. Why?

*Answer:* When you use an assignment operator you are just establishing a relationship between an object and the target. You are merely setting reference to the variable. There are two solutions to this:

(I)

```
>>> a  
[8, 2, 3, 4, 5, 6, 7]  
>>> a = [1,2,3,4,5,6,7]  
>>> b = a[:]  
>>> b[0] = 9  
>>> b  
[9, 2, 3, 4, 5, 6, 7]  
>>> a  
[1, 2, 3, 4, 5, 6, 7]  
>>>
```

(II)

```
>>> a = [1,2,3,4,5,6,7] >>>  
b = list(a)  
>>> b  
[1, 2, 3, 4, 5, 6, 7] >>> b[0]  
= 9  
>>> b  
[9, 2, 3, 4, 5, 6, 7] >>> a  
[1, 2, 3, 4, 5, 6, 7] >>>
```

**Question:** Take a look at the following piece of code:

```
>>> import copy  
>>> a = [1,2,3,4,5,6]  
>>> b = copy.copy(a)  
>>> b  
[1, 2, 3, 4, 5, 6] >>>  
b[2]=9  
>>> b  
[1, 2, 9, 4, 5, 6] >>> a  
[1, 2, 3, 4, 5, 6] >>>
```

'b' is shallow copy of 'a' however when we make changes to 'b' it is not reflected in 'a'. why? How can this be resolved?

*Answer:*

List ‘a’ is an mutable object (list) that consist of immutable objects (integer).

Shallow copy would work with the list containing mutable objects.

You can use  $b=a$  to get the desired result.

**Question: Look at the following code:**

```
>>> my_list = [["apples", "banana"], ["Rose",  
"Lotus"], ["Rice", "Wheat"]]  
>>> copy_list = list(my_list)  
>>> copy_list[2][0] = "cereals"
```

What would happen to content of my\_list? Does it change or remains the same?

*Answer:* Content of my\_list will change:

```
>>> my_list  
[['apples', 'banana'], ['Rose', 'Lotus'],  
['cereals', 'Wheat']]  
>>>
```

**Question: Look at the following code:**

```
>>> my_list = [["apples", "banana"], ["Rose",  
"Lotus"], ["Rice", "Wheat"]]  
>>> copy_list = my_list.copy()  
>>> copy_list[2][0] = "cereals"
```

What would happen to content of my\_list? Does it change or remains the same?

*Answer:* Content of my\_list would change:

```
>>> my_list  
[['apples', 'banana'], ['Rose', 'Lotus'],  
['cereals', 'Wheat']]  
>>>
```

**Question:** When base address of immutable objects are copied it is called \_\_\_\_\_.

*Answer:* Shallow copy

**Question: What happens when nested list undergoes deep copy?**

*Answer:* When we create a deep copy of an object, copies of nested objects in the original object are recursively added to the new object. Thus, deep copy will create complete independent copy of not only of the object but also of its nested objects.

**Question: What happens when nested list undergoes shallow copy?**

*Answer:* A shallow copy just copies references of nested objects therefore the copies of objects are not created.

### Dynamic Arrays

As the name suggests dynamic array is a contiguous block of memory that grows dynamically when new data is inserted. It has the ability to adjust its size automatically as and when there is a requirement, as a result of which, we need not specify the size of the array at the time of allocation and later we can use it to store as many elements as we want.

When a new element is inserted in the array, if there is space then the element is added at the end else a new array is created that is double in size of the current array, so elements are moved from old array to new array and the old array is deleted in order to create some free memory space. The new element is then added at the end of the expanded array.

Let's try to execute a small piece of code. This example is executed on a 32-bit machine architecture. The result can be different from that of 64-bit but the logic remains the same.

For a 32-bit system 32-bits (i.e 4 bytes) are used to hold a memory address.

So, now let's try and understand how this works:

When we created a blank list structure, it occupies 36 bytes of size. Look at the code given as follows:

```
import sys
my_dynamic_list = []
print("length = ",len(my_dynamic_list),".", "size in
bytes = ",sys.getsizeof(my_dynamic_list),".")
```

Here we have imported the sys module so that we can make use of `getsizeof()` function to find the size, the list occupies in the memory.

The output is as follows:

Length = 0.

Size in bytes = 36 .

Now, suppose we have a list of only one element, let's see how much size it occupies in the memory.

```
import sys
my_dynamic_list = []
print("length = ", len(my_dynamic_list), ".",
      "size in bytes = ", sys.getsizeof(my_dynamic_list), ".")
length = 1 . size in bytes = 40 .
```

This 36 bytes is just the requirement of the list data structure on 32-bit architecture.

If the list has one element that means it contains one reference to memory and in a 32-bit system architecture memory, address occupies 4 bytes. Therefore, size of the list with one element is  $36+4 = 40$  bytes.

Now, let's see what happens when we append to an empty list.

```
import sys
my_dynamic_list = []
value = 0
for i in range(20):
    print("i = ", i, ".",
          "Length of my_dynamic_list = ", len(my_dynamic_list), ".",
          "size in bytes = ", sys.getsizeof(my_dynamic_list), ".")
    my_dynamic_list.append(value)
    value += 1
```

Output

i = 0 . Length of my\_dynamic\_list = 0 . size in bytes = 36 . i = 1 . Length of my\_dynamic\_list = 1 . size in bytes = 52 . i = 2 . Length of my\_dynamic\_list = 2 . size in bytes = 52 . i = 3 . Length of my\_dynamic\_list = 3 . size in bytes = 52 . i = 4 . Length of my\_dynamic\_list = 4 . size in bytes = 52 . i = 5 . Length of my\_dynamic\_list = 5 . size in bytes = 68 . i = 6 . Length of my\_dynamic\_list = 6 . size in bytes = 68 . i = 7 . Length of my\_dynamic\_list = 7 . size in bytes = 68 . i = 8 . Length of my\_dynamic\_list = 8 . size in bytes = 68 .

```
i = 9 . Length of my_dynamic_list = 9 . size in bytes = 100 .
i = 10 . Length of my_dynamic_list = 10 . size in bytes = 100 .
i = 11 . Length of my_dynamic_list = 11 . size in bytes = 100 .
i = 12 . Length of my_dynamic_list = 12 . size in bytes = 100 .
i = 13 . Length of my_dynamic_list = 13 . size in bytes = 100 .
i = 14 . Length of my_dynamic_list = 14 . size in bytes = 100 .
i = 15 . Length of my_dynamic_list = 15 . size in bytes = 100 .
i = 16 . Length of my_dynamic_list = 16 . size in bytes = 100 .
i = 17 . Length of my_dynamic_list = 17 . size in bytes = 136 .
i = 18 . Length of my_dynamic_list = 18 . size in bytes = 136 .
i = 19 . Length of my_dynamic_list = 19 . size in bytes = 136 .
```

Now, lets have a look at how things worked here:

When you call an `append()` function for list, resizing takes place as per `list_resize()` function defined in `Objects/listobject.c` file in Python. The job of this function is to allocate cells proportional to the list size thereby making space for additional growth.

The growth pattern is : 0,4,8,16,25,35,46,58,72,88,.....

### Amortization

Let's suppose that there is a man called Andrew, who wants to start his own car repair shop and has a small garage. His business starts and he gets his first customer however, the garage has space only to keep one car. So, he can only have one car in his garage.



Seeing a car in his garage, another person wants to give his car to him. Andrew, for the ease of his business, wants to keep all cars at one place. So, in order to keep two cars, he must look for space to keep two cars, move the old car to the new space and also move the new car to the new space and see how it works.

So, basically he has to:

1. Buy new space
2. Sell the old space

Let's say, this process takes one unit of time.

Now, he also has to:

1. Move old car to new location
2. Move new car to new location

Moving each car takes one unit of time.

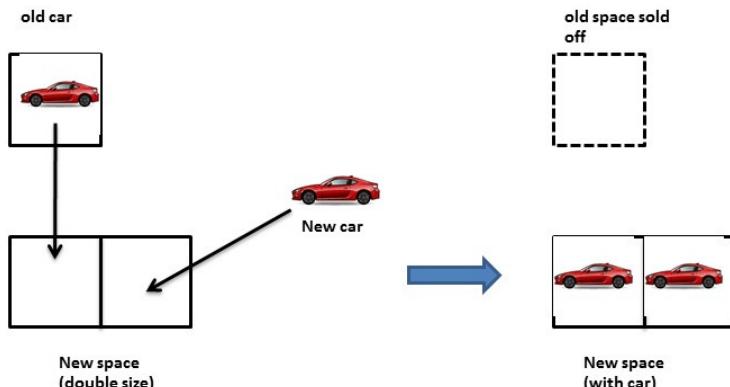


Figure 24

Andrew is new in business. He does not know how his business would expand, also what is the right size for the garage. So, he comes up with the idea that if there is space in his garage then he would simply add the new car to the space and when the space is full he will get new space twice as big as the present one and then move all cars there and get rid of old space. So, the moment he gets his new car, its time to buy a new space twice the old space and get rid of the old space.

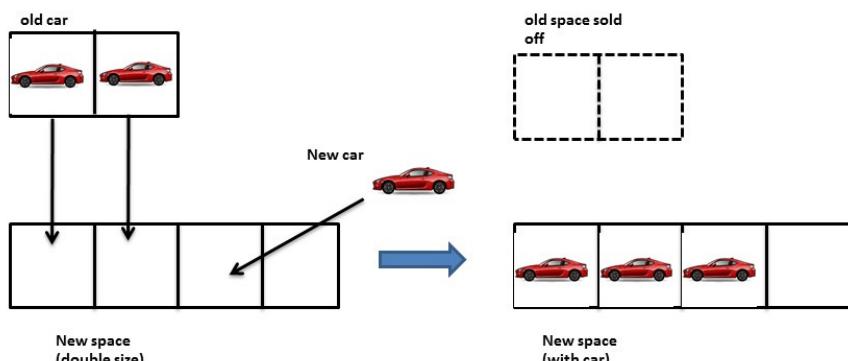
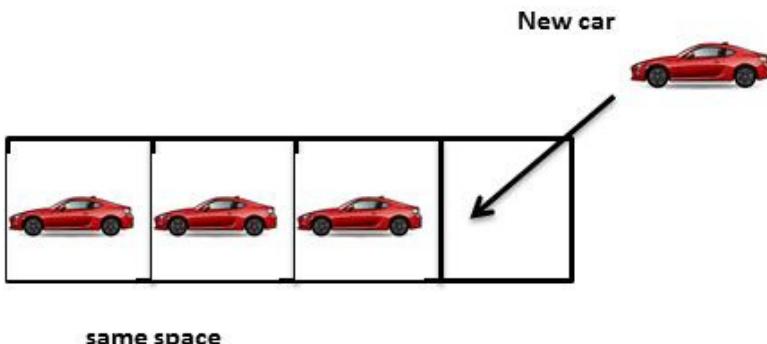


Figure 25

Now, when the fourth car arrives Andrew need not worry .He has space for the car.



*Figure 26*

And now again when he gets the fifth car he will have to buy a space that is double the size of the space that he currently has and sell of the old space.

So, let's now take a look at the time complexity: Let's analyse how much does it takes to add a car to Andrew's garage where there are n number of cars in the garage.

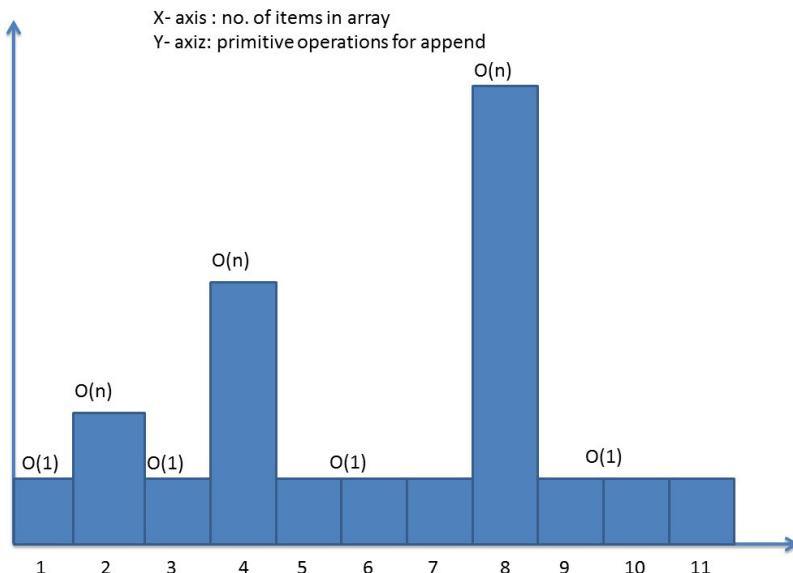
Here is what we have to see:

1. If there is space available, Andrew just has to move one car into new space and that takes only one unit of time. This action is independent of the n (number of cars in the garage). Moving a car in a garage that has space is constant time i.e. O(1).
2. When there is spot and a new car arrives, Andrew has to do the following:
  - a. Buy a new space that takes 1 unit of time.
  - b. Move all cars into new space one by one and then move the new car into free space. Moving every car takes one unit of time. So, if there were n cars already in the old garage, plus one car then that would take  $n+1$  time units to move the car.

So, the total time taken in step two is  $1+n+1$  and in Big O notation this would mean O(n) as the constant does not matter.

On the look of it one may think this is too much of a task but every business plan should be correctly analysed. Andrew will buy a new garage only if the space that he has, gets all filled up. On spreading out the cost over a period of time, one will realize that it takes good amount of time only when the space is full but in a scenario where there is space, addition of cars does not take much of time.

Now keeping this example in mind we try to understand the amortization of dynamic arrays. We have already learnt that in dynamic array when the array is full and new value has to be added, the contents of the array are moved on to the new array that is double in size and then the space occupied by the old array is released.



It may seem like the task of replacing the old array with a new one is likely to slow down the system. When the array is full, appending a new element may require  $O(n)$  time. However, once the new array has been created we can add new elements to the array in constant time  $O(1)$  till it has to be replaced again. We will now see that with the help of amortization analysis how this strategy is actually quite efficient.

As per the graph above, when there are two elements then on calling append, the array will have to double in size, same after 4th and 8th element. So, at 2, 4, 8, 16... append will be O(n) and for the rest of the cases it will be O(1).

Steps involved are as follows:

1. When the array is full, allocate memory for new array and the size of the new array should typically be twice the size of old array.
2. All contents from the old array should be copied to the new array.
3. The space occupied by the old array should be released.

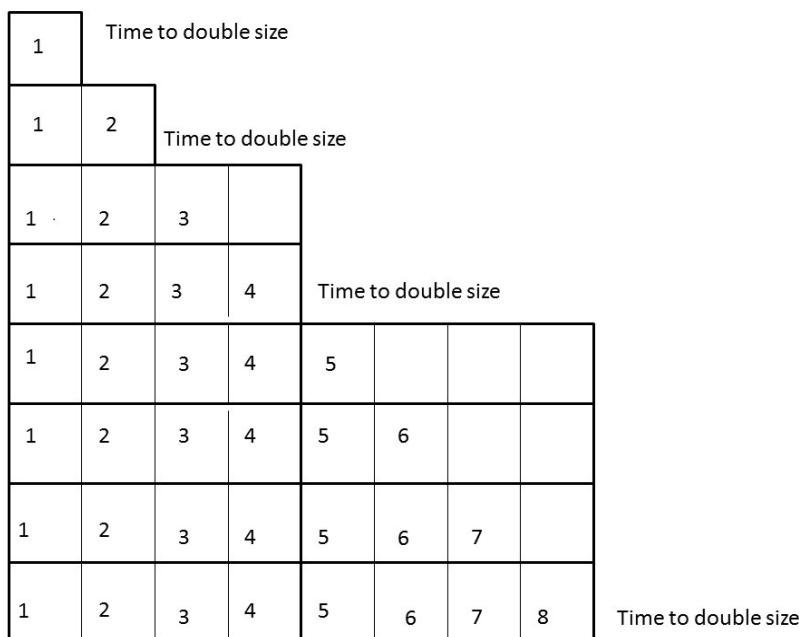


Figure 28

The analysis would be as follows:

| element           | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  | 10 |
|-------------------|---|---|---|---|---|---|---|---|----|----|
| Size of array     | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| cost of insertion | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9  | 1  |

for element 1,4,6,7,8,20.. Cost of insertion is one because we have space to add new element  
 For 2nd element cost of insertion is 2 because we move item 1 and then append the second item  
 Similarly, for item 3 cost of insertion is 3 because we first move two elements from the old array and then append the third item .

$$\text{Amortization cost} = \frac{(1+2+3+1+5+1+1+1+9+1\dots)}{n}$$

Simplify terms greater than 1 as : 2 = 1+1, 3 = 2+1, 5 = 4+1

$$\begin{aligned}\text{Amortization cost} &= \frac{(1+1+1+1+1)+(2+4+6+\dots)}{n} \\ &= \frac{n+2n}{n} \\ &= 3\end{aligned}$$

Hence O(1)

Figure 29

## Chapter 10

# Stacks, Queues, and Deque

Stack, Queues, and Deque are linear structures in Python.

### Stack

Stack is an ordered collection of items where the addition and removal of items take place at the same end which is also known as the TOP. The other end of the stack is known as the BASE. The base of the stack is significant because the items that are closer to the base represent those that have been in the stack, for the longest time.

The most recently added item is the one that is in the top position so that it can be removed first.

This principle of ordering is known as LIFO—that stands for Last-in-first-out, where the newer items are near the top, while older items are near the base.

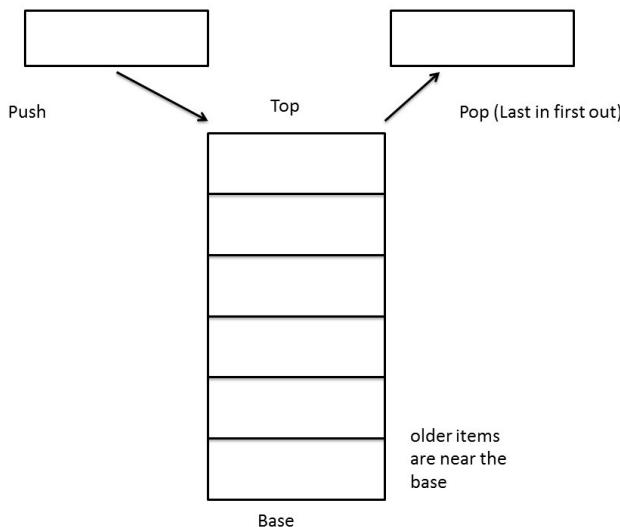


Figure 30

## 150 - Python Interview Questions

Stacks are important because they are required whenever there is a need to reverse the order of items as the order of removal is reverse of the order of insertion.

Example:

1. Pushing back button on browser while surfing on internet.
2. Ctrl+Z (undo) in Microsoft applications.
3. Remove recently used object from cache.

Stacks are commonly used by software programmers; it may not be quite noticeable while using a program as these operations generally take place at the background. However, many times you may come across Stack overflow error. This happens when a stack actually runs out of memory.

Stacks seem to be very simple. Yet they are one of the most important data structures as you will see very soon. Stacks serve as a very important element of several data structures and algorithms.

**Question: Explain, how would you implement a stack in Python.**

*Answer:* We will implement Stack using lists.

Step 1:

Define the Stack Class

```
#Define      Stack  
Class class Stack:
```

Step 2:

Create constructor

Create a constructor that takes self and size (n) of the stack. In the method we declare that self.stack is empty list([]) and self.size is equal to n i.e. the size provided.

```
#Define Stack Class  
def __init__(self, n): self.stack = []  
    self.size = n
```

Step 3:

Define Push Function

A push function will have two arguments self and the element that we want to push in the list. In this function we first check if the length of the

stack is equal to the size provided as input (n). If yes, then it means that the stack is full and prints the message that *no more elements can be appended as the stack is full*. However, if that is not the case then we can call the append method to push the element to the stack.

```
def push(self,element):
    if(len(self.stack)== self.size):
        print("no more elements can be appended
as the stack is full")
    else:
        self.stack.append(element)
```

Step 4:

Define POP Function

Check the stack. If it is empty, then print: *Stack is empty. Nothing to POP!!*  
If it is not empty pop the last item from the stack.

```
def pop(self):
    if self.stack == []:
        print("Stack is empty. Nothing to POP!!")

    else:
        return self.stack.pop()
```

The complete code will be as follows:

Code

```
#Define Stack Class
class Stack:
    #declare constructor

    def __init__(self, n):
        self.stack = []
        self.size = n

    #push operation

    def push(self,element):
        if(len(self.stack)== self.size):
            print("no more elements can be appended as
the stack is full")
        else:
```

```
self.stack.append(element)

#pop operation
def pop(self):
    if self.stack == []:
        print("Stack is empty. Nothing to POP!!")
    else:
        self.stack.pop()
```

### Execution

```
s = Stack(3)
s.push(6)
s.push(2)
print(s.stack)
s.pop()
print(s.stack)
```

### Output

```
[6, 2]
[6]
>>>
```

**Question:** Write a program to check if a given string has balanced set of parenthesis or not.

Balanced parenthesis: (), {}, [], {[()]}, [][], etc.

*Answer:*

Here we have to check if pairs of brackets exist in right format in a string. Expressions such as “[]{()}" are correct. However, if the opening bracket does not find the corresponding closing bracket then the parenthesis is a mismatch. For example: “[{}” or “{}})”. To solve this problem we will follow following steps:

Step 1

Define class parenthesis\_match

```
class parenthesis_match:
```

Step 2

Defining lists for opening and closing brackets

We now define two lists such that the index of an opening bracket matches with the index of the corresponding closing bracket:

1. List opening\_brackets will have all types of opening brackets as elements as elements – [“(”, “{”, “[“]

2. List closing\_brackets will have all types of closing brackets as elements – [“)”, “}”, “]”]

Here is how we define the lists:

```
opening_brackets = ["(", "{", "["]
closing_brackets = [")", "}", "]"]
```

Step 3

Defining Constructor, Push, and Pop functions

Constructor:

The constructor takes expression as parameter which is the string provided for validation of parameters.

We also initialize list for stacking purposes. The push() and pop() functions will be applied on this list.

```
def __init__(self, expression):
    self.expression = expression
    self.stack = []
```

Push() and Pop() Functions push() pop

Since we are implementing this using a stack it is quite obvious that we would require and functions.

The push() function when called, will add element to the stack.

```
def push(self, element):
    self.stack.append(element)
```

The pop() element on the other hand will pop the last element from the stack.

```
#pop operation
def pop(self):
    if self.stack == []:
        print("Unbalanced Paranthesis")
    else:
        self.stack.pop()
```

Step 4

Defining the function to do the analysis

We will now write the code to analyse the string.

In this function we will perform the following steps:

- First we check the length of the expression. A string of balanced parenthesis will always have even number of characters. So, if the length of the expression is divisible by two only then we would move ahead with the analysis. So, an if...else loop forms the outer structure of this function.

```
if len(self.expression)%2 == 0:  
    ---- we analyse.....  
else:  
    print("Unbalanced Paranthesis")
```

- Now, considering that we have received a length of even number. We can move ahead with analysis and we can write our code in the “if” block. We will now traverse through the list element by element. If we encounter an opening bracket we will push it on to the stack, if it is not an opening bracket then we check if the element is in the closing bracket list. If yes then we pop the last element from the stack and see if the index of the elements in opening\_brackets and closing\_brackets list is of same bracket. If yes, then there is a match else the list is unbalanced.

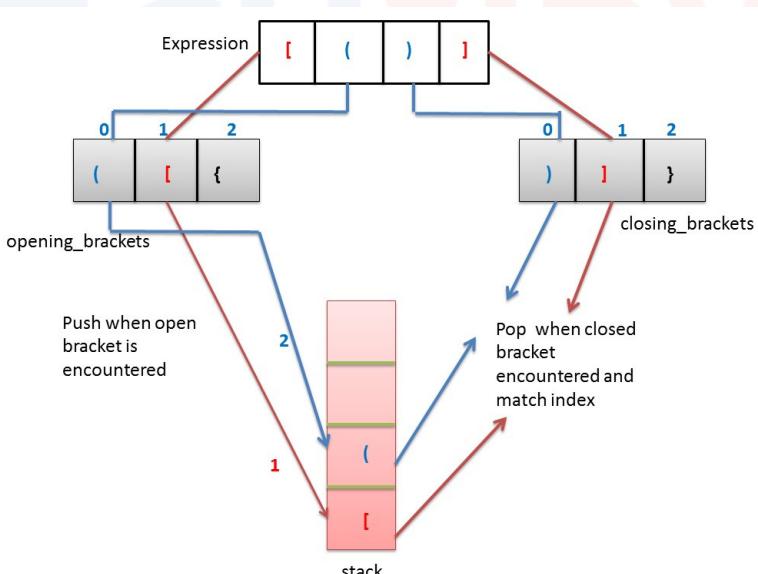


Figure 31

```
if element in self.opening_brackets:  
    self.push(element)  
elif element in self.closing_brackets:  
    x = self.stack.pop()  
    if self.opening_brackets.index(x) == self.  
        closing_brackets.index(element):  
            print("Match Found")  
        else:  
            print("Match not found - check prnthesis")  
    return;
```

- So, the final code will be as follows, to make things easier for the end user, print commands have been added:

```
class parenthesis_match:  
    opening_brackets = ["(", "{", "["]  
  
    closing_brackets = [")", "}", "]"]  
  
    #declare constructor  
    def __init__(self, expression):  
        self.expression = expression  
        self.stack = []  
  
    #push operation  
    def push(self, element):  
        self.stack.append(element)  
  
    #pop operation  
    def pop(self):  
        if self.stack == []:  
            print("Unbalanced Paranthesis")  
        else:  
            self.stack.pop()  
    def is_match(self):  
        print("expression is = ", self.expression)  
        if len(self.expression)%2 == 0:  
            for element in self.expression:  
                print("evaluating ", element)  
                if element in self.opening_brackets: print("it is an opening  
bracket - ", element, "pushing to stack")
```

## 156 - Python Interview Questions

```
self.push(element)
print("pushed", element, " on to stack the stack is ",
self.stack)
elif element in self.closing_brackets:
x = self.stack.pop()
print("time to pop element is ", x)
if self.opening_brackets.index(x)==
self.closing_brackets.index(element):
print("Match Found")
else:
print("Match not found - check prarnthesis")
return;
else:
print("Unbalanced Parenthesis")
```

### Execution

```
pm = parenthesis_match("([{}])")
pm.is_match()
```

### Output

```
expression is = ([{}])
evaluating (
it is an opening bracket - ( pushing to stack
pushed ( on to stack the stack is ['(']
evaluating [
it is an opening bracket - [ pushing to stack
pushed [ on to stack the stack is ['(', '[']
evaluating {
it is an opening bracket - { pushing to stack
pushed { on to stack the stack is ['(', '[',
'{']
evaluating }
time to pop element is {
Match Found
evaluating ]
time to pop element is [
Match Found
evaluating )
time to pop element is (
Match Found
```

## Queue

A queue is a sequence of objects where elements are added from one end and removed from the other. The queues follow the principle of first in first out. The removal of items is done from one end called the *Front* and the items are removed from another end that's referred to as *rear*. So, just as in case of any queue in real life, the items enter the queue from the rear and start moving towards the front as the items are removed one by one.

So, in Queue the item at the front is the one that has been in the sequence for the longest and the most recently added item must wait at the end. The insert and delete operations are also called enqueue and dequeue.

Basic Queue functions are as follows:

- `enqueue(i)` : Add element “i” to the queue.
- `dequeue()` : Removes the first element from the queue and returns its value.

$\Sigma$  `isEmpty()` : Boolean function that returns “true” if the queue is empty else it will return false.

$\Sigma$  `size()` : Returns length of the queue.



Figure 32

**Question: Write a code for implementation of Queue.**

**Answer:** The implementation of queue is as follows:

Step1

Define the class

class

Queue: Step

Define the constructor

Here, we initialize an empty list queue:

```
def __init__(self):
    self.queue = []
```

### Step 3

Define isEmpty() function

As the name suggests, this method is called to check if the queue is empty or not. The function check the queue. If it is empty, it prints a message - *Queue is Empty* or else it will print a message - *Queue is not Empty*

```
def isEmpty(self):
    if self.queue == []:
        print("Queue is Empty")
    else:
        print("Queue is not Empty")
```

### Step 4

Define enqueue() function

This function takes an element as parameter and inserts it at index “0”. All elements in the queue shift by one position.

```
def enqueue(self,element):
    self.queue.insert(0,element)
```

### Step 5

Define dequeue() function

This function pops the oldest element from the queue.

```
def dequeue(self):
    self.queue.pop()
```

### Step 6

Define size() function

This function returns the length of the queue.

```
def size(self):
    print("size of queue is",len(self.queue))
```

### Code

```
class Queue:  
    def __init__(self):  
        self.queue = []  
    def isEmpty(self):  
        if self.queue == []:  
            print("Queue is Empty")  
        else:  
            print("Queue is not empty")  
    def enqueue(self,element):  
        self.queue.insert(0,element)  
    def dequeue(self):  
        self.queue.pop()  
    def size(self):  
        print("size of queue is",len(self.queue))
```

### Code Execution

```
q = Queue()  
q.isEmpty()  
print("inserting element no.1")  
q.enqueue("apple")  
print("inserting element no.2")  
q.enqueue("banana")  
print("inserting element no.3")  
q.enqueue("orange")  
print("The queue elements are as follows:")  
print(q.queue)  
print("check if queue is empty?") q.isEmpty()  
print("remove first element")  
q.dequeue()  
print("what is the size of the queue?")  
q.size()  
print("print contents of the queue")  
print(q.queue)
```

## Output

```
Queue is Empty
inserting element no.1
inserting element no.2
inserting element no.3
The queue elements are as follows:
['orange', 'banana', 'apple']
check if queue is empty?
Queue is not empty
remove first element
what is the size of the queue?
size of queue is 2
print contents of the queue
['orange', 'banana']
```

**Question:** Write a code to implement a stack using single queue. What is the time complexity for *push()* and *pop()* functions?

**Answer:** Before implementing the code it is important to understand the logic behind it. The question demands that you make a queue work like a stack. A queue works on the principle of First-In-First-Out whereas a stack works on the principle of Last In First Out.

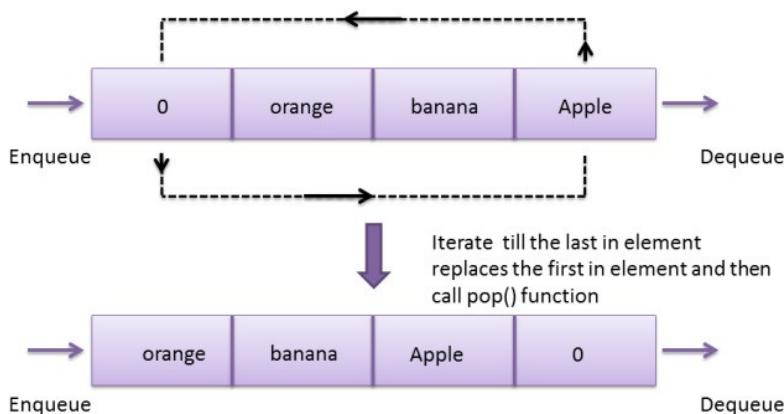


Figure 33

### Code

```
class Stack_from_Queue:  
    def __init__(self):  
        self.queue = []  
    def isEmpty(self):  
        if self.queue == []:  
            print("Queue is Empty")  
        else:  
            print("Queue is not empty")  
    def enqueue(self,element):  
        self.queue.insert(0,element)  
    def dequeue(self):  
        return self.queue.pop()  
    def size(self):  
        print("size of queue is",len(self.queue))  
    def pop(self):  
        for i in range(len(self.queue)-1):  
            x = self.dequeue()  
            print(x)  
            self.enqueue(x)  
        print("element removed is",self.dequeue())
```

### Execution – I

```
sq = Stack_from_Queue()  
sq.isEmpty()  
print("inserting element apple")  
sq.enqueue("apple")  
print("inserting element banana")  
sq.enqueue("banana")  
print("inserting element orange")  
sq.enqueue("orange")  
print("inserting element O") sq.enqueue("O")  
print("The queue elements are as follows:")  
print(sq.queue)  
print("check if queue is empty?")  
sq.isEmpty()  
print("remove the last element") sq.pop()  
print(sq.queue)
```

### Output – I

```
Queue is Empty
inserting element apple
inserting element banana
inserting element orange
inserting element 0
The queue elements are as follows:
['0', 'orange', 'banana', 'apple']
check if queue is empty?
Queue is not empty
remove the last in element
apple
banana
orange
element removed is 0
['orange', 'banana', 'apple']
```

### Execution – II

```
sq = Stack_from_Queue()
sq.isEmpty()
print("inserting element apple")
sq.enqueue("apple")
print("inserting element banana")
sq.enqueue("banana")
print("inserting element orange")
sq.enqueue("orange")
print("inserting element 0")
sq.enqueue("0")
for i in range(len(sq.queue)):
    print("The queue elements are as follows:")
    print(sq.queue)
    print("check if queue is empty?")
    sq.isEmpty()
    print("remove the last in element")

print(sq.queue)
```

## Output -II

```
Queue is Empty
inserting element apple
inserting element banana
inserting element orange
inserting element 0
The queue elements are as follows:
['0', 'orange', 'banana', 'apple']
check if queue is empty?
Queue is not empty
remove the last in element
apple
banana
orange
element removed is 0
['orange', 'banana', 'apple']
The queue elements are as follows:
['orange', 'banana', 'apple'] check if
queue is empty?
Queue is not empty
remove the last in element
apple
banana
element removed is orange
['banana', 'apple']
The queue elements are as follows:
['banana', 'apple']
check if queue is empty?
Queue is not empty
remove the last in element
apple
element removed is banana
['apple']
The queue elements are as follows:
['apple']
check if queue is empty?
Queue is not empty
remove the last in element
element removed is apple
[]
>>>
```

Time complexity for push() and pop() function is O(1)

Time complexity for push is O(1).

Time complexity for pop is O(n) because we have to iterate through the loop and rearrange elements before retrieving the element.

**Question: How can a queue be implemented using two stacks?**

*Answer:*

Step 1:

Create a basic Stack() class with push(), pop(), and isEmpty() functions.

```
class Stack:  
    def __init__(self):  
        self.stack = []  
  
    def push(self,element):  
        self.stack.append(element)  
  
    def pop(self):  
        return self.stack.pop()  
    def isEmpty(self):  
        return self.stack == []
```

Step 2:

Define the Queue class

```
class Queue:
```

Step 3:

Define the constructor

Since the requirement here is of two stacks, we initialize two stack objects.

```
def __init__(self):  
    self.inputStack = Stack()  
    self.outputStack = Stack()
```

Step 4:

Define the enqueue function

This function will push the elements into the first stack.

```
def enqueue(self,element):
    self.inputStack.push(element)
```

Step 5:

Define the dequeue() function

This function checks if the output stack is empty or not. If it is empty then elements will be popped out from inputStack one by one and pushed into the outputStack, so that the last in element is the first one to be out. However, if the outputStack is not empty, then the elements can be popped directly from it.

Now suppose we insert 4 values: 1,2,3,4 calling the enqueue function.  
Then the input stack would be like this:

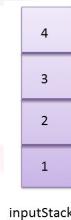


Figure 34

When we call dequeue function, the elements from inputStack are popped and pushed one by one on to the output stack till we reach the last element and that last element is popped from the inputStack and returned. If the output stack is not empty then it means that it already has elements in right order and they can be popped in that order.

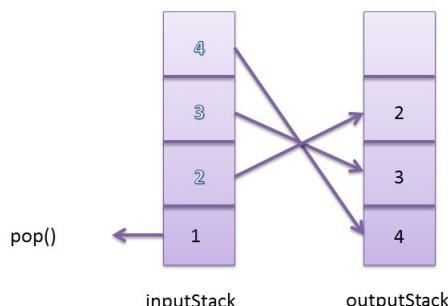


Figure 35

```
def dequeue(self):
    #if not self.inputStack.isEmpty():
    if self.outputStack.isEmpty():
        for i in range(len(self.inputStack.stack)-1):
            x = self.inputStack.pop()
            self.outputStack.push(x)
            print("popping out value =", self.
inputStack.pop())
        else:
            print("popping out value =", self.
outputStack.pop())
```

### Code

```
class Queue:
    def __init__(self):
        self.inputStack = Stack()
        self.outputStack = Stack()
    def enqueue(self,element):
        self.inputStack.push(element)
    def dequeue(self):
        if self.outputStack.isEmpty():
            for i in range(len(self.inputStack.stack)-1):
                x = self.inputStack.pop()
                self.outputStack.push(x)
                print("popping out value =", self.
inputStack.pop())
            else:
                print("popping out value =", self.
outputStack.pop())

#Define Stack Class
class Stack:
    def __init__(self):
        self.stack = []
    def push(self,element):
        self.stack.append(element)

    def pop(self):
```

```
return self.stack.pop()

def isEmpty(self):
    return self.stack == []
```

### Execution

```
Q = Queue()
print("insert value 1")
Q.enqueue(1)
print("insert value 2")
Q.enqueue(2)
print("insert value 3")
Q.enqueue(3)
print("insert value 4")
Q.enqueue(4)
print("dequeue operation")
Q.dequeue()
Q.dequeue()
print("insert value 7")
Q.enqueue(7)
Q.enqueue(8)
Q.dequeue()
Q.dequeue()
Q.dequeue()
Q.dequeue()
```

### Output

```
Q = Queue()
print("insert value 1")
Q.enqueue(1)
print("insert value 2")
Q.enqueue(2)
print("insert value 3")
Q.enqueue(3)
print("insert value 4")
Q.enqueue(4)
print("dequeue operation")
Q.dequeue()
Q.dequeue()
print("insert value 7")
Q.enqueue(7)
Q.enqueue(8)
```

```

Q.dequeue()
Q.dequeue()
Q.dequeue()
Q.dequeue()

```

### Deques

A deque is more like a queue only that it is double ended. It has items positioned in ordered collection and has two ends, the front and the rear. A deque is more flexible in nature in the sense that the elements can be added or removed from front or rear. So you get the qualities of both stack and queue in this one linear data structure.

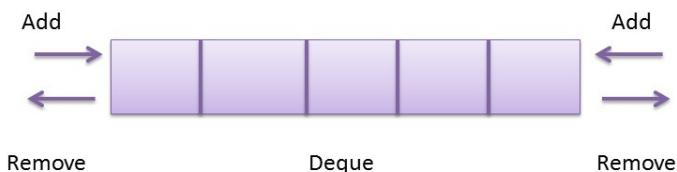


Figure 36

### Question: Write a code to implement a deque.

**Answer:** Implementation of deque is easy. If you have to add an element from rear, you will have to add it at index 0 same way if you have to add it from the front, call the append() function. Similarly, if you have to remove front call pop() function and if you want to pop it from the rear call .

pop(0)

### Code

```

class Deque:
    def __init__(self):
        self.deque = []
    def addFront(self,element):
        self.deque.append(element)
        print("After adding from front the deque value is :",
              self.deque)
    def addRear(self,element):
        self.deque.insert(0,element)

    print("After adding from end the deque value is :",
          self.deque)
    def removeFront(self):
        self.deque.pop()

```

```
print("After removing from the front the deque  
value is : ", self.deque)  
def removeRear(self):  
    self.deque.pop(0)  
    print("After removing from the end the deque  
value is : ", self.deque)
```

### Execution

```
D = Deque()  
print("Adding from front")  
D.addFront(1)  
print("Adding from front")  
D.addFront(2)  
print("Adding from Rear")  
D.addRear(3)  
print("Adding from Rear")  
D.addRear(4)  
print("Removing from Front")  
D.removeFront()  
print("Removing from Rear")  
D.removeRear()
```

### Output

```
After adding from front the deque value is : [1] After  
adding from front the deque value is : [1, 2]  
After adding from end the deque value is : [3, 1, 2]  
After adding from end the deque value is : [4, 3, 1, 2]  
After removing from the front the deque value is : [4,  
3, 1]  
After removing from the end the deque value is : [3, 1]  
>>>
```

# Chapter 11

## Linked List

Linked list is a linear structure that consists of elements such that each element is an individual object and contains information regarding:

1. Data
2. Reference to next element

In linked list, each element is called a node.

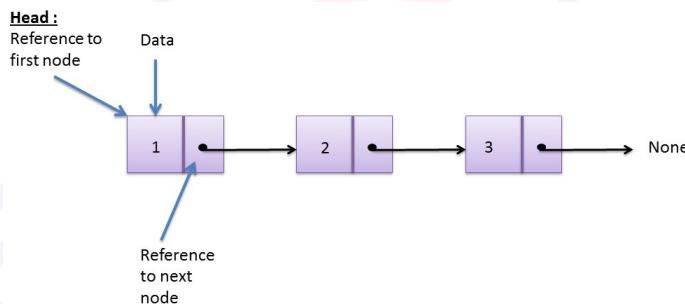


Figure 37

You can see in the diagram, the reference to the first node is called *Head*. It is the entry point of the linked list. If the list is empty then the head points to null. The last node of the linked list refers to null.

As the number of nodes can be increased or decreased as per requirement, linked lists are considered to be dynamic data structure. However, in a linked list direct access to data is not possible. Search for any item starts from the head and you will have to go through each reference to get that item. A linked list occupies more memory.

The linked list described above is called a singly linked list. There is one more type of linked list known as doubly linked list. A double linked list has reference to the next node and the previous node.

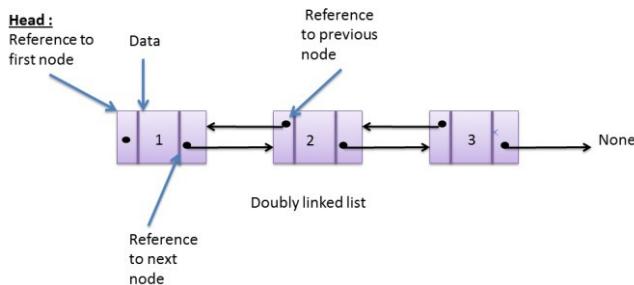


Figure 38

**Question:** Write a code to implement a *Node* class such that a *Node* contains data as well as a reference to the next node.

*Answer:*

To create a node object we pass data value to the constructor. The constructor assigns the data value to data and sets node's reference to *None*. Once we create all the objects we assign memory address of the second object as the reference to the first node object, memory address of third object is assigned as reference to second object, and so on. The last object, thus have no(or none as) reference.

The code for the Node class will be as follows:

Code

```
class Node:  
    def __init__(self,data = None):  
        self.data = data  
        self.reference = None  
objNode1 = Node(1)  
objNode2 = Node(2)  
objNode3 = Node(3)  
objNode4 = Node(4)  
objNode1.reference = objNode2  
objNode2.reference = objNode3  
objNode3.reference = objNode4  
objNode4.reference = None
```

## Execution

```
print("DATA VALUE = ",objNode1.data,"REFERENCE = ",objNode1.reference)
print("DATA VALUE = ",objNode2.data,"REFERENCE = ",objNode2.reference)
print("DATA VALUE = ",objNode3.data,"REFERENCE = ",objNode3.reference)
print("DATA VALUE = ",objNode4.data,"REFERENCE = ",objNode4.reference)
```

## Output

```
DATA VALUE = 1 REFERENCE = <__main__.Node object at 0x000000595E0CC6A0>
DATA VALUE = 2 REFERENCE = <__main__.Node object at 0x000000595E329978>
DATA VALUE = 3 REFERENCE = <__main__.Node object at 0x000000595E3299B0>
DATA VALUE = 4 REFERENCE = None
>>>
```

### Question: Write code to traverse through a linked list.

*Answer:*

#### Method I

We have already written the code for the Node class:

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.reference = None

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
```

We will now see how to traverse through the linked list:

**Step 1**

We create a variable `presentNode` and assign the first object to it.

```
presentNode = objNode1
```

On doing this `presentNode` gets the data and reference values of `objNode`

### 1 Step 2

The reference value points to `objNode2`.

So, we can write a while loop:

```
while presentNode:  
    print("DATA VALUE = ",presentNode.data)  
    presentNode = presentNode.reference
```

Once the `presentNode` is assigned, the reference value contained in `objNode4` in it will come out of the while loop because the value of reference is *None*.

Code

```
class Node:  
    def __init__(self,data = None):  
        self.data = data  
        self.reference = None
```

Execution

```
objNode1 = Node(1)  
objNode2 = Node(2)  
objNode3 = Node(3)  
objNode4 = Node(4)  
objNode1.reference = objNode2  
objNode2.reference = objNode3  
objNode3.reference = objNode4  
objNode4.reference = None  
presentNode = objNode1  
while presentNode:  
    print("DATA VALUE = ",presentNode.data)  
    presentNode = presentNode.reference
```

Output

```
DATA VALUE =  
1 DATA VALUE  
= 2 DATA  
VALUE = 3  
DATA VALUE =  
4
```

## Method II

Another method to do this by creating two classes: Node and Linked list  
Code

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.reference = None
class Linked_list:
    def __init__(self):
        self.head = None
    def traverse(self):
        presentNode = self.head
        while presentNode:
            print("DATA VALUE = ",presentNode.data)
            presentNode = presentNode.reference
```

## Execution

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
# reference of the first node object to second object

linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.traverse()
```

## Output

```
DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4
```

**Question:** Write a code to add a node at the beginning of a linked list.

**Answer:** To solve this question, we will just add a new method to insert the node in the same code that is mentioned in the last example:

In the last example, we pointed the head of the linked list object to the first node object.

```
linkObj.head = objNode1
```

When we add the node at the beginning we just have to make the linkObj.head = new\_node and new\_node.reference = obj\_Node1.

For this we write a code where, the value of the linkObj.head is first passed on to new\_node.reference and then linkObj.head is set to the new node object.

```
def insert_at_Beginning(self,data):
    new_data = Node(data)
    new_data.reference = self.head
    self.head = new_data
```

So, the full code would be like the following:

Code

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.reference = None
class Linked_list:
    def __init__(self):
        self.head = None
    def traverse(self):
        presentNode = self.head
        while presentNode:
            print("DATA VALUE = ",presentNode.data)
            presentNode = presentNode.reference

    def insert_at_Beginning(self,data):
        new_data = Node(data)
        new_data.reference = self.head
        self.head = new_data
```

## Execution

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
# reference of the first node object to second
object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.insert_at_Beginning(5)
linkObj.traverse()

```

## Output

```

DATA  VALUE  =  5
DATA  VALUE  =  1
DATA  VALUE  =  2
DATA  VALUE  =  3
DATA VALUE = 4

```

### **Question: Write a code to add a node at the end of a linked list.**

*Answer:* In order to add a node at the end of the linked list it is important that you point the reference of the last node to the new node that you create.

Step 1:

Define the function

```
def insert_at_end(self,data):
```

Step 2:

Create a new Node object

```
new_data = Node(data)
```

Step 3:

Traverse through the linked list to reach the last node

Remember that you cannot directly access the last node in the linked list. You will have to traverse through all the nodes and reach the last node in order to take the next step.

```
presentNode = self.head
while presentNode.reference != None:
    presentNode = presentNode.reference
```

Step 4:

Add the new node at the end

After traversing through the linked list, you know that you have reached the last node when *presentNode.reference = None*. Since this won't remain the last node anymore, you need to:

```
presentNode.reference = new_data
```

With this we have added a new node at the end of a linked list.

Code

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.reference = None
class Linked_list:
    def __init__(self):
        self.head = None
    def traverse(self):
        presentNode = self.head
        while presentNode:
            print("DATA      VALUE      =",presentNode.data)
            presentNode = presentNode.reference

    def insert_at_end(self,data):
        new_data = Node(data)
        presentNode = self.head
        while presentNode.reference != None:
            presentNode = presentNode.reference
        presentNode.reference = new_data
```

## Execution

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
# reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.insert_at_end(5)
linkObj.insert_at_end(6)
linkObj.insert_at_end(7)
linkObj.traverse()

```

## Output

```

DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4
DATA VALUE = 5
DATA VALUE = 6
DATA VALUE = 7

```

**Question:** Write a code to insert a node between two nodes in a linked list.

**Answer:** The solution for this problem is very similar to adding a node to the beginning. The only difference is that when we add a node at the beginning we point the head value to the new node whereas in this case the function will take two parameters. First will be the node object after which the new object will be inserted and second would be the data for the new object. Once the new node is created, we pass on the reference value stored in the existing node object to it and the existing node is then made to point at the new node object

Step 1:

Define the function

This function will take two parameters:

## 180 - Python Interview Questions

1. The node object after which the data is to be inserted.
2. Data for the new node object.

```
def insert_in_middle(self,insert_data,new_data):
```

Step 2:

Assign references

```
new_node = Node(new_data)
new_node.reference = insert_data.reference
insert_data.reference = new_node
```

Code

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.reference = None
class Linked_list:
    def __init__(self):
        self.head = None
    def traverse(self):
        presentNode = self.head
        while presentNode:
            print("DATA VALUE = ",presentNode.data)
            presentNode = presentNode.reference

    def insert_in_middle(self,insert_data,new_data):
        new_node = Node(new_data)
        new_node.reference = insert_data.reference
        insert_data.reference = new_node
```

Execution

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
# reference of the first node object to second object
linkObj.head.reference = objNode2
```

```

objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.insert_in_middle(objNode3,8)
linkObj.traverse()

```

Output

```

DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 8
DATA VALUE = 4
>>>

```

**Question: Write code to remove a node from a linked list.**

*Answer:* Suppose we have a linked list as shown below:

A -> B -> C

A.reference = B

B.reference = C

C.reference = A.

To remove B, we traverse through the linked list. When we reach node A that has reference pointing to B, we replace that value with the reference value stored in B(that points to C) . So that will make A point to C and B is removed from the chain.

The function code will be as follows:

```

def remove(self,removeObj):
    presentNode = self.head
    while presentNode:
        if(presentNode.reference == removeObj):
            presentNode.reference = removeObj.reference
            presentNode = presentNode.reference

```

The function takes the Node object as parameter and traverses through the linked list, till it reaches the object that needs to be removed. Once we reach the node that has reference to the node that has to be removed, we simply change the reference value to reference value stored in object removeObj.. Thus, the node now points directly to the node after the removeObj.

### Code

```
class Node:  
    def __init__(self,data = None):  
        self.data = data  
        self.reference = None  
class Linked_list:  
    def __init__(self):  
        self.head = None  
    def traverse(self):  
        presentNode = self.head  
        while presentNode:  
            print("DATA      VALUE      =",presentNode.data)  
            presentNode = presentNode.reference  
  
    def remove(self,removeObj):  
        presentNode = self.head  
        while presentNode:  
            if(presentNode.reference == removeObj):  
                presentNode.reference = removeObj.reference  
            presentNode = presentNode.reference
```

### Execution

```
objNode1 = Node(1)  
objNode2 = Node(2)  
objNode3 = Node(3)  
objNode4 = Node(4)  
linkObj = Linked_list()  
#head of the linked list to first object  
linkObj.head = objNode1  
# reference of the first node object to second  
# object  
linkObj.head.reference = objNode2  
objNode2.reference = objNode3  
objNode3.reference = objNode4  
linkObj.remove(objNode2)  
linkObj.traverse()
```

## Output

```
DATA VALUE =
1 DATA VALUE
= 3 DATA
VALUE = 4
```

**Question:** Print the values of the node in the center of the linked list.

**Answer:** This involves counting the number of nodes in the object. If the length is even then the data of two nodes in the middle should be printed else only the data of node in the center should be printed.

Step 1:

Define the function

```
def find_middle(self,llist):
```

Step 2:

Find the length of the counter

Here, we set a variable counter = 0. As we traverse through the linked list we increment the counter. At the end of the while loop we get the count of number of nodes in the linked list. This is also the length of the linked list.

```
counter = 0
presentNode = self.head
while presentNode:
    presentNode = presentNode.reference  counter =
    counter + 1
    print("size of linked list = ",counter)
```

Step 3:

Reach the middle of the linked list

The reference to the node in the middle is stored in the node before that. So, in the for loop instead of iterating (counter/2) times, we iterate (counter-1)/2 times. This brings us to the node which is placed just before the centre value.

```
presentNode = self.head
for i in range((counter-1)//2):
    presentNode = presentNode.reference
```

Step 4:

Display the result depending on whether the number of nodes in the linked list

If the linked list has even number of nodes then print the value of reference stored in the present node and the next node.

```
if (counter%2 == 0):
    nextNode = presentNode.reference
    print("Since the length of linked list is an even
          number the two middle elements are:")
    print(presentNode.data,nextNode.data)
```

Else, print the value of the present node.

```
else:
    print("Since the length of the linked list is an odd
          number, the middle element is: ")
    print(presentNode.data)
```

Code

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.reference = None
class Linked_list:
    def __init__(self):
        self.head = None
    def find_middle(self,llist):
        counter = 0
        presentNode = self.head
        while presentNode:
            presentNode = presentNode.reference  counter =
            counter + 1
            print("size of linked list = ",counter)
            presentNode = self.head
            for i in range((counter-1)//2):
                presentNode = presentNode.reference
```

```

if (counter%2 == 0):
    nextNode = presentNode.reference
    print("Since the length of linked list is an even
number the two middle elements are:")
    print(presentNode.data,nextNode.data)

else:
    print("Since the length of the linked list is an odd
number, the middle element is: ")
    print(presentNode.data)

```

### Execution (Odd Number of Nodes)

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
objNode5 = Node(5)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
# reference of the first node object to second
object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
objNode4.reference = objNode5
linkObj.find_middle(linkObj)

```

### Output

```

size of linked list = 5
Since the length of the linked list is an odd
number, the middle element is:
3

```

### Execution (Even Numbers)

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
# reference of the first node object to second
object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.find_middle(linkObj)
```

### Output

```
size of linked list = 4
Since the length of linked list is an even number the
two middle elements are:
2 3
```

### Question: Implement doubly linked list.

Answer: A doubly linked list has three parts:

1. Pointer to the previous node
2. Data
3. Pointer to the next node

Implementation of doubly linked list is easy. We just need to take care of one thing that each node is connected to the next and the previous data.

Step 1:

Create a Node Class

The node class will have a constructor that initializes three parameters: data, reference to next node – refNext and reference to previous node – refPrev.

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.refNext = None
        self.refPrev = None
```

### Step 2:

Create functions to traverse through the double linked list I.

#### Traverse Forward

To traverse forward with the help of refNext that points to next value of the linked list. We start with the head and move on to the next node using refNext.

```
def traverse(self):
    presentNode = self.head
    while presentNode:
        print("DATA    VALUE    =    ",presentNode.data)
        presentNode = presentNode.refNext
```

#### II. Traverse Reverse

Traverse reverse is opposite of traverse forward. We are able to traverse backwards with the help of value of refPrev because it points to previous node. We start from the tail and move on to the previous node using refPrev.

```
def traverseReverse(self):
    presentNode = self.tail
    while presentNode:
        print("DATA    VALUE    =    ",presentNode.data)
        presentNode = presentNode.refPrev
```

### Step 3:

Write a function to add a node at the end

Appending a node at the end of the doubly linked list is same as appending in linked list the only difference is that we have to ensure that the node that is appended has its refPrev pointing to the node after which it has been added.

```
def append(self,data):
    new_data = Node(data)
    presentNode = self.head
    while presentNode.refNext != None:
        presentNode    =    presentNode.refNext
    presentNode.refNext = new_data
    new_data.refPrev = presentNode
```

### Step 4:

Write function to remove a node

This function takes the node object that needs to be removed as parameter. In order to remove a node we iterate through the doubly linked list twice. We first start with the head and move forward using refNext and when we encounter the object that needs to be removed we change the refNext value of the present node (which is presently pointing to the object that needs to be removed) to the node that comes after the object that needs to be removed. We then traverse through the linked list backwards starting from tail and when we encounter the object to be removed again we change the refPrev value of the present node to the node that is placed before it.

```
def remove(self,removeObj):
    presentNode = self.head
    presentNodeTail = self.tail
    while presentNode.refNext != None:
        if(presentNode.refNext == removeObj):
            presentNode.refNext = removeObj.refNext
            presentNode = presentNode.refNext
        while presentNodeTail.refPrev != None:
            if(presentNodeTail.refPrev == removeObj):
                presentNodeTail.refPrev = removeObj.refPrev
                presentNodeTail = presentNodeTail.refPrev
```

### Code

```
class Node:
    def __init__(self,data = None):
        self.data = data
        self.refNext = None
        self.refPrev = None
class dLinked_list:
    def __init__(self):
        self.head = None
        self.tail = None
    def append(self,data):
        new_data = Node(data)
        presentNode = self.head
        while presentNode.refNext != None:
            presentNode = presentNode.refNext
```

```
presentNode.refNext = new_data
new_data.refPrev = presentNode
self.tail = new_data

def traverse(self):
    presentNode = self.head
    while presentNode:
        print("DATA VALUE = ",presentNode.data)
        presentNode = presentNode.refNext

def traverseReverse(self):
    presentNode = self.tail
    while presentNode:
        print("DATA VALUE = ",presentNode.data)
        presentNode = presentNode.refPrev
    def remove(self,removeObj):
        presentNode = self.head
        presentNodeTail = self.tail
        while presentNode.refNext != None:
            if(presentNode.refNext == removeObj):
                presentNode.refNext = removeObj.refNext
                presentNode = presentNode.refNext
            while presentNodeTail.refPrev != None:
                if(presentNodeTail.refPrev == removeObj):
                    presentNodeTail.refPrev = removeObj.refPrev
                    presentNodeTail = presentNodeTail.refPrev
```

### Execution

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
dlinkObj = dLinked_list()
#head of the linked list to first object
dlinkObj.head      =
objNode1.dlinkObj.tail =
#objNode1 reference of the first node object to second
object
dlinkObj.head.refNext = objNode2
dlinkObj.tail.refPrev = objNode3
objNode2.refNext = objNode3
objNode3.refNext = objNode4
objNode4.refPrev = objNode3
objNode3.refPrev = objNode2
objNode2.refPrev = objNode1
print("Appending Values")
dlinkObj.append(8)
dlinkObj.append(9)
print("traversing forward after Append")
dlinkObj.traverse()
print("traversing reverse after Append")
dlinkObj.traverseReverse()
print("Removing Values")
dlinkObj.remove(objNode2)
print("traversing forward after Remove")
dlinkObj.traverse()
print("traversing reverse after Remove")
dlinkObj.traverseReverse()
```

## Output

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
dlinkObj = dLinked_list()
#head of the linked list to first object
dlinkObj.head      =
objNode1.dlinkObj.tail =
#objNode1 reference of the first node object to second
object
dlinkObj.head.refNext = objNode2
dlinkObj.tail.refPrev = objNode3
objNode2.refNext = objNode3
objNode3.refNext = objNode4
objNode4.refPrev = objNode3
objNode3.refPrev = objNode2
objNode2.refPrev = objNode1

print("Appending Values")
dlinkObj.append(8)
dlinkObj.append(9)
print("traversing forward after Append")
dlinkObj.traverse()
print("traversing reverse after Append")
dlinkObj.traverseReverse()
print("Removing Values")
dlinkObj.remove(objNode2)
print("traversing forward after Remove")
dlinkObj.traverse()
print("traversing reverse after Remove")
dlinkObj.traverseReverse()

```

**Question: Write code to reverse a linked list.**

**Answer:** To reverse a linked list we have to reverse the pointers. Look at the figure shown below. The first table shows how information is stored in the linked list. The second table shows how the parameters are initialized in the reverse() function before beginning to traverse through the list and reversing the elements.

## 192 - Python Interview Questions

| Node 1                |                     | Node 2          |              | Node 3      |              | Node 4 |              |
|-----------------------|---------------------|-----------------|--------------|-------------|--------------|--------|--------------|
| data                  | reference to        | data            | reference to | data        | reference to | data   | reference to |
| 1                     | node2               | 2               | node3        | 3           | node4        | 4      | none         |
| <b>Initialization</b> |                     |                 |              |             |              |        |              |
| Parameters            |                     | set to value of |              | Final Value |              |        |              |
| previous              | None                | None            |              | presentNode | self.head    | node1  |              |
| nextval               | presentNode.refNext | node2           |              |             |              |        |              |

Figure 39

We then use the following while loop:

```
while nextval != None:
    presentNode.refNext = previous
    previous = presentNode
    presentNode = nextval
    nextval = nextval.refNext
    presentNode.refNext = previous
    self.head = presentNode
```

This is how the while loop works:

The diagram illustrates the state transition of variables through three nodes (node1, node2, node3) during a while loop iteration. It shows three tables for each node, representing the 'While Loop' state, followed by a table showing the final values after the loop iteration.

| While Loop          |                 |
|---------------------|-----------------|
| Parameter           | Set To          |
| presentNode.refNext | previous        |
| previous            | presentNode     |
| presentNode         | nextval         |
| nextval             | nextval.refNext |

| Parameter           | Value          |
|---------------------|----------------|
| presentNode.refNext | refers to None |
| previous            | node1          |
| presentNode         | node2          |
| nextval             | node3          |

| While Loop          |                 |
|---------------------|-----------------|
| Parameter           | Set To          |
| presentNode.refNext | previous        |
| previous            | presentNode     |
| presentNode         | nextval         |
| nextval             | nextval.refNext |

| Parameter           | Value           |
|---------------------|-----------------|
| presentNode.refNext | refers to node1 |
| previous            | node2           |
| presentNode         | node3           |
| nextval             | node4           |

| While Loop          |                 |
|---------------------|-----------------|
| Parameter           | Set To          |
| presentNode.refNext | previous        |
| previous            | presentNode     |
| presentNode         | nextval         |
| nextval             | nextval.refNext |

| Parameter           | Value           |
|---------------------|-----------------|
| presentNode.refNext | refers to node1 |
| previous            | node2           |
| presentNode         | node3           |
| nextval             | node4           |

Figure 40

You can see as we iterate through the while loop how the values of presentnode.refNext change. Node1 that was earlier pointing to node2 changes its pointer to none. Same way node2 changes its pointer value to node1, and so on.

## Code

```

class Node:
    def __init__(self,data = None):
        self.data = data
        self.refNext = None
class Linked_list:
    def __init__(self):
        self.head = None

    def reverse(self):
        previous = None
        presentNode = self.head
        nextval = presentNode.refNext
        while nextval != None:
            presentNode.refNext = previous
            previous = presentNode
            presentNode = nextval
            nextval = nextval.refNext
            presentNode.refNext = previous
        self.head = presentNode
    def traverse(self):
        presentNode = self.head
        while presentNode:
            print("DATA      VALUE      =      ",presentNode.data)
            presentNode = presentNode.refNext

```

## Execution

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()

#head of the linked list to first object
linkObj.head = objNode1

# reference of the first node object to second object
linkObj.head.refNext = objNode2
objNode2.refNext = objNode3
objNode3.refNext = objNode4
print("traverse before reversing")
linkObj.traverse()
linkObj.reverse()
print("traverse after reversing")
linkObj.traverse()

```

Output

```
traverse before reversing
DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4
traverse after reversing
DATA VALUE = 4
DATA VALUE = 3
DATA VALUE = 2
DATA VALUE = 1
```



# Chapter 12

## Recursion

When a function makes a call to itself it is called recursion. The same sets of instructions are repeated again and again for new values and it is important to decide when the recursive call must end. For Example: Let's look at the code to find factorial of a number.

If we use a loop then a factorial function would look something like this:

Code

```
def factorial(number):
    j = 1
    if number==0|number==1:
        print(j)
    else:
        for i in range (1, number+1):
            print(j," * ",i," = ",j*i)  j = j*i
        print(j)
```

Execution

```
factorial(5)
```

Output

```
1 * 1 = 1
1 * 2 = 2
2 * 3 = 6
6 * 4 = 24
24 * 5 = 120
120
>>>
```

Now, let's have a look at how we can solve the same problem using a recursive algorithm.

### Code

```
def factorial(number):
    j = 1
    if number==0|number==1:
        return j
    else:
        return number*factorial(number-1)
```

### Execution

```
print(factorial(4))
```

### Output

24

### Pros and Cons

Recursive functions make the code look neat, it helps in breaking a complex task into a simpler sub-problems. It can be easier than implementing iterations. However, it can be little difficult to understand the logic behind recursion. Recursion can consume more memory and time and can be hard to debug.

Question: Write code to find the sum of natural numbers from 0 to the given number using recursion.

Answer:

| i | Result                    |
|---|---------------------------|
| 0 | 0                         |
| 1 | $1 + 0 = i(1) + i(0) = 1$ |
| 2 | $+ 1 = i + i(1) = 3$      |
| 3 | $3 + 3 = i + i(2) = 6$    |
| 4 | $4 + 6 = i + i(3) = 10$   |
| 5 | $5 + 10 = i + i(4) = 15$  |

Observe for  $i = 0$ , the result is 0, thereafter result =  $i(n)+i(n-1)$

### Code

```
def natural_sum(num):
    if num == 0:
        return 0
    else:
        return (num + natural_sum(num-1))
```

## Execution

```
print(natural_sum(10))
```

## Output

```
55
```

**Question:** What would be the output for the following code?

```
def funny(x,y):
    if y == 1:
        return x[0]
    else:
        a = funny(x, y-1)  if a > x[y-1]:
            return a
        else:
            return x[y-1]
x = [1,5,3,6,7]
y = 3
print(funny(x,y))
```

## Answer:

If we insert print statement in the code and execute the code again we can see the actual sequence in which it executes:

```
def funny(x,y):
    print("calling funny , y = ",y)
    if y == 1:
        return x[0]
    else:
        print("inside else loop because y = ", y)
        a = funny(x, y-1)
        print("a = ",a)
        if a > x[y-1]:
            print("a = ",a, " Therefore a > ",x[y-1])  return a
        else:
            print("a = ",a, " Therefore a < ",x[y-1])  return x[y-1]
x = [1,5,3,6,7]
y = 3
print(funny(x,y))
```

## 198 - Python Interview Questions

### Output

```
calling funny , y = 3
inside else loop because y = 3
calling funny , y = 2
inside else loop because y = 2
calling funny , y = 1
a = 1
a = 1 Therefore a < 5
a = 5
a = 5 Therefore a > 3
5
The answer is 5
```

### Question: What would be the output of the following code?

```
def funny(x):
    if (x%2 == 1):
        return x+1
    else:
        return funny(x-1)
print(funny(7))
print(funny(6))
```

#### Answer:

For x = 7

1. x = 7
2. x % 2 is 1
- return 7 + 1

For x = 6

1. x = 6
2. x%2 = 0
3. Return funny(5)
4. x = 5
5. x%2 = 1
6. Return x+1 = 6

### Question: Write Fibonacci sequence using recursion.

#### Answer:

The Fibonacci sequence = 0, 1, 2, 3, 5, 8, 13.....

| i | Result                  |
|---|-------------------------|
| 0 | 0                       |
| 1 | 1                       |
| 2 | $1+0 = i(0) + i(1) = 1$ |
| 3 | $1+1 = i(2) + i(1) = 2$ |
| 4 | $2+1 = i(3) + i(2) = 3$ |
| 5 | $3+2 = i(4) + i(3) = 5$ |

Observe for  $i = 0$ , the result is 0 and for  $i = 1$ , the result is 1. Thereafter, the value of  $i(n) = i(n-1) + i(n-2)$ . We implement the same, when we try to find Fibonacci code using recursion.

- The fibonacci\_seq(num), takes a number as argument.
- If num = 0, result is 0
- If num = 1, result is 1
- Else result is fibonacci\_seq(num-1) + Fibonacci\_seq(num-2)
- If you want to find Fibonacci Sequence for 10 then:
  - o For elements 0 to 10
  - o Call the fibonacci\_seq() function

fibonacci\_seq(0) = 0  
 fibonacci\_seq(1) = 1  
 fibonacci\_seq(2) = fibonacci\_seq(1)+ fibonacci\_seq(0)  
 fibonacci\_seq(3) = fibonacci\_seq(2)+ fibonacci\_seq(3)

### Code

```

def fibonacci_seq(num):
  if num <0:
    print("Please provide a positive integer value")
  if num == 0:
    return 0
  elif num == 1:
    return 1
  else:
    return (fibonacci_seq(num-1)+fibonacci_seq(num-2))
  
```

## 200 - Python Interview Questions

### Execution

```
for i in range(10):
    print(fibonacci_seq(i))
```

### Output

```
0
1
1
2
3
5
8
13
21
34
```

### Question: What is memoization?

**Answer:** Basically in memoization we maintain a look up table where solutions are stored so that we don't have to solve the same sub problem again and again. Instead we solve it once and store the values so that they can be reused.

We know that Fibonacci sequence is:

$$F(n) = F(n-1)+F(n-2) \text{ if } n>1
= n \text{ if } n=0,1$$

So,

```
F(n):
if n<1:
    return n
else :
    return F(n-1)+F(n-2)
```

Here, we are making two recursive calls and adding them up and the value is returned.

Look at the following diagram:

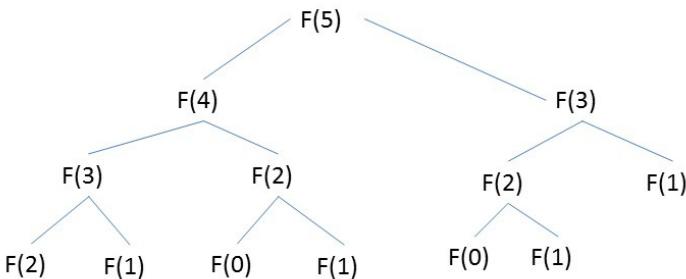


Figure 41

Just to find fibonacci(5), fibonacci(2) is computed three times and fibonacci(3) is computed two times. So, as  $n$  increases fibonacci function's performance will go down. The consumption of time and space would increase exponentially with increase in  $n$ . In order to save time what we can do is save a value when it is computed for the first time. So, we can save  $F(2)$  when it is computed for the first time, same way with  $F(3), F(4) \dots$  so on. So, we can say that:

```

F(n):
if n<=1:
    return n
elif F(n) exist : return
    F(n-1)
else:
    F(n) = F(n-1)+F(n-2)
    Save F(n).
    Return F(n).
  
```

In the code below:

1. The function `fibonacci()` takes a number and creates a list, `fib_num` of size `num+1`. This is because the Fibonacci series start from 0.
2. It calls the function `fib_calculate()` which takes the number `num` and list `fib_num` as parameter.
3. We have saved -1 at all index in the list:
  - a. If `fib_num[num]` is  $>0$ , that means Fibonacci for this number already exists and we need not compute it again and the number can be returned.

## 202 - Python Interview Questions

- b. If num <= 1 then return num.
- c. Else if num >=2, calculate fib\_calculate(num - 1, fib\_num) + fib\_calculate(num - 2, fib\_num). The value calculated must be stored in list fib\_num at index num so that there is no need to calculate it again.

Code

```
def fibonacci(num):  
  
    fib_num = [-1]*(num + 1)  
    return fib_calculate(num, fib_num)  
def fib_calculate(num, fib_num):  
    if fib_num[num] >= 0:  
        return fib_num[num]  
  
    if (num <= 1):  
        fnum = num  
    else:  
        fnum = fib_calculate(num - 1, fib_num) + fib_  
        calculate(num - 2, fib_num)  
        fib_num[num] = fnum  
  
    return fnum
```

Execution

```
num = int(input('Enter the number: '))  
print("Answer = ",fibonacci(num))
```

Output

```
Enter the number: 15  
Answer = 610  
=>
```

**Question: What would be the output of the following program?**

```
def test_function(i,j):  
    if i == 0:  
        return j;  
    else:  
        return test_function(i-1,j+1)  
print(test_function(6,7))
```

*Answer:*

| i | j  | i == 0 ? | return              |
|---|----|----------|---------------------|
| 6 | 7  | No       | test_function(5,8)  |
| 5 | 8  | No       | test_function(4,9)  |
| 4 | 9  | No       | test_function(3,10) |
| 3 | 10 | No       | test_function(2,11) |
| 2 | 11 | No       | test_function(1,12) |
| 1 | 12 | No       | test_function(0,13) |
| 0 | 13 | Yes      | 13                  |

The output will be 13.

**Question: What will be the output for the following code:**

```
def even(k):
    if k <= 0:
        print("please enter a positive value")  elif k == 1:
        return 0
    else:
        return even(k-1) + 2
print(even(6))
```

*Answer:*

| k | k <= 0 | k == 1 | result            |
|---|--------|--------|-------------------|
| 6 | no     | no     | even(5)+2         |
| 5 | no     | no     | Even(4)+2+2       |
| 4 | no     | no     | Even(3)+2+2+2     |
| 3 | no     | no     | Even(2)+2+2+2+2   |
| 2 | no     | no     | Even(1)+2+2+2+2+2 |
| 1 | no     | ye     | 0+2+2+2+2+2 = 10  |

S

**Question: Write a code to find nth power of 3 using recursion.**

*Answer:*

1. Define function n\_power(n), it takes the value of power as parameter(n).
2. If n = 0 then return 1 because any number raise to power 0 is 1.
3. Else return (n\_power(n-1)).

204 - *Python Interview Questions*

| n | n < 0 | n == 0 | result                          |
|---|-------|--------|---------------------------------|
| 4 | No    | no     | <code>n_power(3)*3</code>       |
| 3 | No    | no     | <code>n_power(2)*3*3</code>     |
| 2 | No    | no     | <code>n_power(1)*3*3*3</code>   |
| 1 | No    | no     | <code>n_power(0)*3*3*3*3</code> |
| 0 | No    | ye     | <code>1*3*3*3*3</code>          |

## Code

```
def n_power(n):
    if n < 0:
        print("please enter a positive value")
    elif n == 0:
        return 1
    else:
        return n_power(n-1)*3
```

## Execution

```
print(n**power(4))
```

## Output

81

# Chapter 13

## Trees

There are several types of data structures that can be used to tackle application problems. We have seen how linked lists work in a sequential manner, we have also seen how stacks and queues can be used in programming applications but they are very restricted data structures. The biggest problem while dealing with linear data structure is that if we have to conduct a search, the time taken increases linearly with the size of data. Whereas there are some cases where linear structures can be really helpful but the fact remains that they may not be a good choice for situations that require high speed.

So, now let's move on from the concept of linear data structures to nonlinear data structures called trees. Every tree has a distinguished node called the root. Unlike the trees that we know in real life the tree data structure branches downwards from parent to child and every node except the root is connected by a directly edge from exactly one other node.

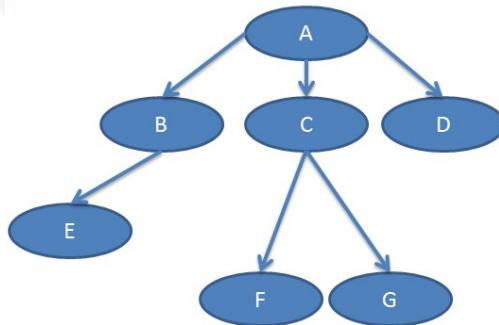


Figure 42

Look at the figure given above:

A is the root and it is parent to three nodes – B, C, and D.

Same way B is parent to E and C is parent to F and G.

Nodes like D, E, F, and G that have no children are called leaves or external nodes.

Nodes that have at least child such as B and C are called internal nodes.

The number of edges from root to node is called the depth or level of a node. Depth of B is 1 whereas depth of G is 2.

Height of a node is the number of edges from the node to the deepest leaf.

B, C, and D are siblings because they have same parent A. Similarly, F and G are also siblings because they have same parent C.

Children of one node are independent of children of another node.

Every leaf node is unique.

Σ The file system that we use on our computer machines is an example of tree structure.

Σ Additional information about the node is known as payload. Payload is not given much of importance in algorithms but it plays a very important role in modern day computer applications.

Σ An edge connects two nodes to show that there is a relationship between them.

Σ There is only one incoming edge to every node (except the root). However, a node may have several outgoing edges.

Σ Root of the tree is the only node of the tree that has no incoming edges as it marks the starting point for a tree.

Σ Set of nodes having incoming edges from the same node are the children of that node.

Σ A node is a parent of all the nodes that connect to it by the outgoing edges.

Σ A set of nodes and edges that comprises of a parent along with all its descendants are called subtrees.

Σ A unique path traverses from the root to each node.

Σ A tree having maximum of two children is called a binary tree.  
Recursive definition of a tree

A tree can be empty, or it may have a root with zero or more subtree.

Root of every subtree is connected to the root of a parent tree by an edge.

### Simple Tree Representation

Have a look at the following tree. Here, we are considering case of binary tree.

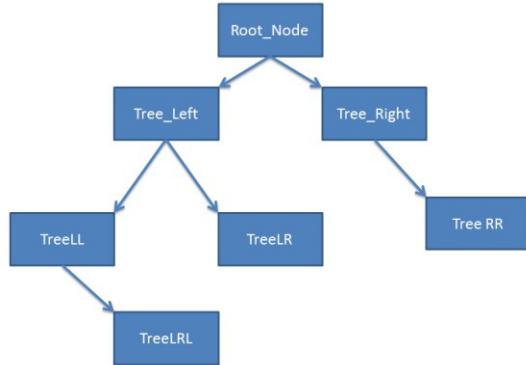


Figure 43

In case of a binary tree a node cannot have more than two children. So, for ease of understanding, we can say that above scenario is similar to something like this:

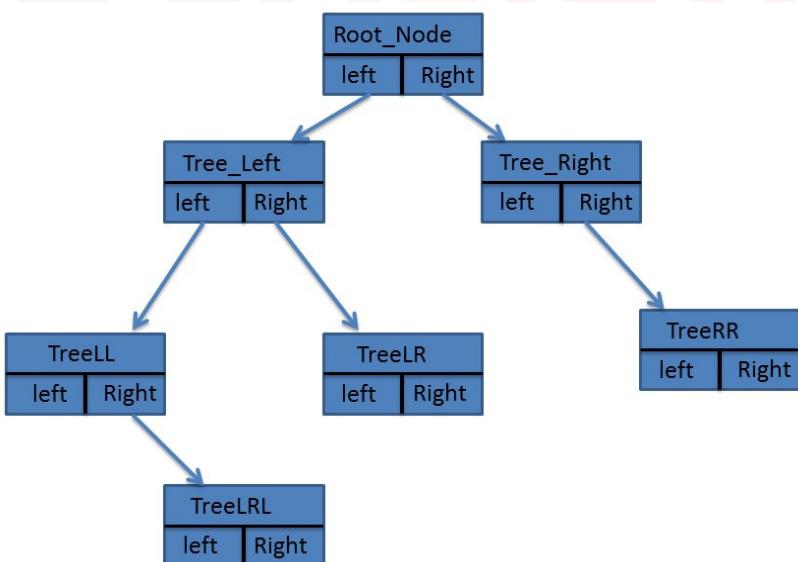


Figure 44

In this diagram, left and right are references to the instances of node that are located on the left and right of this node respectively. As you can see, every node has three values:

1. Data
2. Reference to child on left
3. Reference to child on right

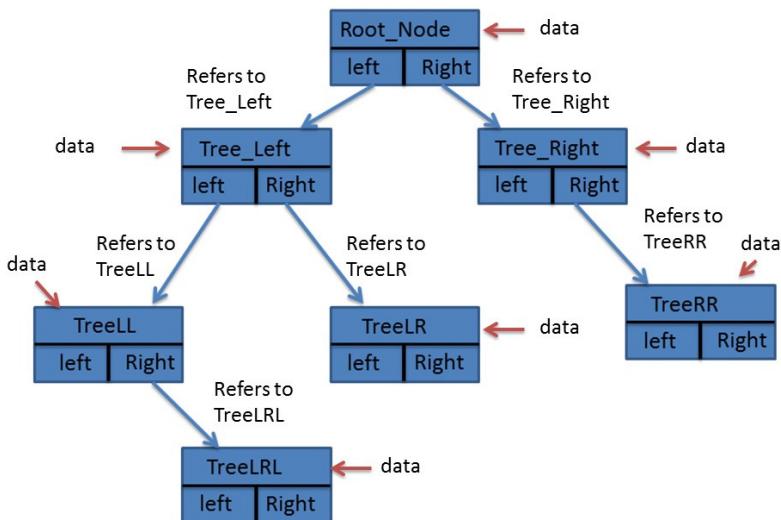


Figure 45

So, the constructor for the node class to create a Node object will be as follows:

```

class Node(object):
    def __init__(self, data_value):
        self.data_value = data_value
        self.left = None
        self.right = None
  
```

So, this is how a root node is created:

```

# Root_Node
print("Create Root Node")
root = Node("Root_Node")
print("Value of Root = ", root.data_value, " left = ", root.left, " right = ", root.right)
  
```

When we execute this block of code, the output is as follows:

Create Root Node

*Value of Root = Root\_Node left = None right = None*

*Value of Node = Tree\_Left left = None right = None*

Now, we write code for inserting values to the left or right.

When a node is created, initially its left and right reference point to None.

So, to add a child to left we just need to say:

`self.left = child_node`

and a child can be added to right in the similar way:

`self.left = child_node`

However, if the root node is already pointing to some existing child and we try to insert a child node then the existing child should be pushed down one level and the new object must take its place. So, the reference of the existing child stored in `self.left` is passed on to `child.left` and then `self.left` is assigned the reference to `child`. This can be achieved in the following manner:

```
def insert_left(self, child):
    if self.left is None:
        self.left = child
    else:
        child.left = self.left
        self.left = child
def insert_right(self, child):
    if self.right is None:
        self.right = child
    else:
        child.right = self.right
        self.right = child
```

Code

```
class Node(object):
    def __init__(self, data_value):
        self.data_value = data_value
        self.left = None
        self.right = None
```

```
def insert_left(self, child):
    if self.left is None:
        self.left = child
    else:
        child.left = self.left
        self.left = child
def insert_right(self, child):
    if self.right is None:
        self.right = child
    else:
        child.right = self.right
        self.right = child
```

### Execution

```
# Root_Node
print("Create Root Node")
root = Node("Root_Node")
print("Value of Root = ",root.data_value," left = ",root.left, " right = ",root.right)
#Tree_Left
print("Create Tree_Left")
tree_left = Node("Tree_Left")
root.insert_left(tree_left)
print("Value of Node = ",tree_left.data_value," left = ",tree_left.left, " right = ",tree_left.right)
print("Value of Root = ",root.data_value," left = ",root.left, " right = ",root.right)
#Tree_Right
print("Create Tree_Right")
tree_right = Node("Tree_Right")
root.insert_right(tree_right)
print("Value of Node = ",tree_right.data_value," left = ",tree_right.left, " right = ",tree_right.right)
print("Value of Root = ",root.data_value," left = ",root.left, " right = ",root.right)
#TreeLL
print("Create TreeLL")
treell = Node("TreeLL")
```

```

tree_left.insert_left(treell)
print("Value of Node = ",treell.data_value," left "
      =",treell.left, " right = ",treell.right)
print("Value of Node = ",tree_left.data_value," left "
      =",tree_left.left, " right = ",tree_left.
      right)
print("Value of Root = ",root.data_value," left "
      =",root.left, " right = ",root.right)

```

**Output**

```

Create Root Node
Value of Root = Root_Node left = None right = None
Create Tree_Left
Value of Node = Tree_Left left = None right = None
Value of Root = Root_Node left = <__main__.Node
object at 0x000000479EC84F60> right = None
Create Tree_Right
Value of Node = Tree_Right left = None right = None
Value of Root = Root_Node left = <__main__.Node
object at 0x000000479EC84F60> right = <__main__.
Node object at 0x000000479ED05E80>
Create TreeLL
Value of Node = TreeLL left = None right =
None
Value of Node = Tree_Left left = <__main__.Node
object at 0x000000479ED0F160> right = None
Value of Root = Root_Node left = <__main__.Node
object at 0x000000479EC84F60> right = <__main__.
Node object at 0x000000479ED05E80>

```

**Question: What is the definition of a tree?**

*Answer:* A tree is a set of nodes that store elements. The nodes have parent- child relationship such that:

- If the tree is not empty, it has a special node called the root of tree. The root has no parents.

- Every node of the tree different from the root has a unique parent node.

**Question:** Write a code to represent a tree through lists or as list of lists.

*Answer:* In list of lists, we shall store the value of the node as the first element. The second element will be the list that represents the left subtree and the third element represents the right subtree. The following figure shows a tree with just the root node.

[‘Root\_Node’, [], []]

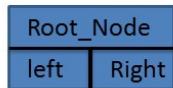


Figure 46

Now, suppose we add a node to the left.

[‘Root\_Node’, [‘Tree\_Left’, [], []], []]

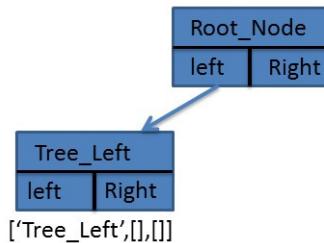


Figure 47

Now, adding another subtree to the right would be equal to:

[‘Root\_Node’, [‘Tree\_Left’, [], []], [‘Tree\_Right’, [], []]]

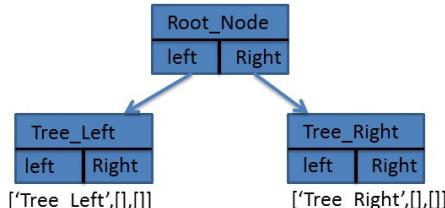


Figure 48

Same way adding a node to the left of Tree\_Left would mean:

### Implement Trees with Lists

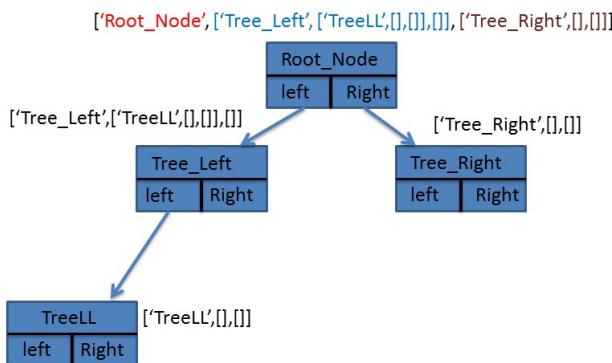


Figure 49

Here you can see that the tree can be defined as follows:

```
binary_tree = ['Root_Node', ['Tree_Left',
['TreeLL', [], []], []], ['Tree_Right', [], []]]
```

Here, the root is Root\_Node which is located at binary\_tree[0].

Left subtree is at binary\_tree[1].

Right subtree is at binary\_tree[2].

Now let's write the code for this.

Step 1:

Define Class

```
class Tree:
```

Step 2:

Create Constructor

Now let's write the code for constructor. Here, when we create an object we pass a value. The constructor creates a list where the value is placed at index 0 and at index 1 and 2 we have two empty list. If we have to add subtree at the left side we will do so at index 1 and for right subtree we will insert values in the list at index 2.

```
def __init__(self,data):
    self.tree = [data, [], []]
```

### Step 3:

Define function to insert left and right subtree

If you have to insert a value in left sub tree then pop the element at index 1 and insert the new list at that place. Similarly, if you have to insert a child at right hand side, pop the value at index 2 and insert the new list.

```
def left_subtree(self,branch):
    left_list = self.tree.pop(1)
    self.tree.insert(1,branch.tree)

    def right_subtree(self,branch):
        right_list = self.tree.pop(2)
        self.tree.insert(2,branch.tree)
```

Now, let's execute the code:

### Code

```
class Tree:
    def __init__(self,data):
        self.tree = [data, [], []]

    def left_subtree(self,branch):
        left_list = self.tree.pop(1)
        self.tree.insert(1,branch.tree)

        def right_subtree(self,branch):
            right_list = self.tree.pop(2)
            self.tree.insert(2,branch.tree)
```

### Execution

```
print("Create Root Node")
root = Tree("Root_node")
print("Value of Root = ",root.tree)
print("Create Left Tree")
tree_left = Tree("Tree_Left")
root.left_subtree(tree_left)
print("Value of Tree_Left = ",root.tree)
print("Create Right Tree")
tree_right = Tree("Tree_Right")
root.right_subtree(tree_right)
print("Value of Tree_Right = ",root.tree)
```

## Output

```
Create Root Node
Value of Root = ['Root_node', [], []]
Create Left Tree
Value of Tree_Left = ['Root_node', ['Tree_Left', [], []], []]
Create Right Tree
Value of Tree_Right = ['Root_node', ['Tree_Left', [], []],
['Tree_Right', [], []]]
Create TreeLL
Value of Tree_Left = ['Root_node', ['Tree_Left',
['TreeLL', [], []], []], ['Tree_Right', [], []]] >>>
```

There is however one thing ignored in this code. What if we want to insert a child somewhere in between? Here, in this case the child will be inserted at the given location and the subtree existing at that place will be pushed down.

For this we make changes in the insert functions.

```
def left_subtree(self,branch):
    left_list = self.tree.pop(1)
    if len(left_list) > 1:
        branch.tree[1]=left_list
        self.tree.insert(1,branch.tree) else:
            self.tree.insert(1,branch.tree)
```

If we have to insert a child in the left then first we pop the element at index 1. If the length of element at index 1 is 0 then, we simply insert the list. However, if the length is not zero then we push the element to the left of the new child. The same happens in case of right subtree.

```
def right_subtree(self,branch):
    right_list = self.tree.pop(2)
    if len(right_list) > 1:
        branch.tree[2]=right_list
        self.tree.insert(2,branch.tree) else:
            self.tree.insert(2,branch.tree) print("Create
TreeLL")
treell = Tree("TreeLL")
tree_left.left_subtree(treell)
print("Value of Tree_Left = ",root.tree)
```

## 216 - Python Interview Questions

### Code

```
class Tree:  
    def __init__(self,data):  
        self.tree = [data, [], []]  
  
    def left_subtree(self,branch):  
        left_list = self.tree.pop(1)  
        if len(left_list) > 1:  
            branch.tree[1] = left_list  
            self.tree.insert(1,branch.tree) else:  
            self.tree.insert(1,branch.tree)  
  
    def right_subtree(self,branch):  
        right_list = self.tree.pop(2)  
        if len(right_list) > 1:  
            branch.tree[2] = right_list  
            self.tree.insert(2,branch.tree) else:  
            self.tree.insert(2,branch.tree)
```

### Execution

```
print("Create Root Node")  
root = Tree("Root_node")  
print("Value of Root = ",root.tree)  
print("Create Left Tree")  
tree_left = Tree("Tree_Left")  
root.left_subtree(tree_left)  
print("Value of Tree_Left = ",root.tree)  
print("Create Right Tree")  
tree_right = Tree("Tree_Right")  
root.right_subtree(tree_right)  
print("Value of Tree_Right = ",root.tree)  
print("Create Left Inbetween")  
tree_inbtw = Tree("Tree left in between")  
root.left_subtree(tree_inbtw)  
print("Value of Tree_Left = ",root.tree)  
print("Create TreeLL")  
treell = Tree("TreeLL")  
tree_left.left_subtree(treell)  
print("Value of TREE = ",root.tree)
```

## Output

```
Create Root Node
Value of Root = ['Root_node', [], []]
Create Left Tree
Value of Tree_Left = ['Root_node', ['Tree_Left', [], []], []]
Create Right Tree
Value of Tree_Right = ['Root_node', ['Tree_Left', [], []],
['Tree_Right', [], []]]
Create Left Inbetween
Value of Tree_Left = ['Root_node', ['Tree left in
between', ['Tree_Left', [], []], []], ['Tree_Right', [], []]]
Create TreeLL
Value of TREE = ['Root_node', ['Tree left in between',
['Tree_Left', ['TreeLL', [], []], []], ['Tree_Right', [], []]]]
>>>
```

**Question: What is a binary tree? What are the properties of a binary tree?**

**Answer:** A data structure is said to be binary tree if each of its node can have at most two children. The children of the binary tree are referred to as the left child and the right child.

**Question: What are tree traversal methods?**

**Answer:**

### Traversals

**Preorder Traversal:** First visit the root node, then visit all nodes on the left followed by all nodes on the right.

### Code

```
class Node(object):
    def __init__(self, data_value):
        self.data_value = data_value
        self.left = None
        self.right = None
    def insert_left(self, child):
        if self.left is None:
            self.left = child
```

```
else:  
    child.left = self.left  
    self.left = child  
def insert_right(self, child):  
    if self.right is None:  
        self.right = child  
    else:  
        child.right = self.right  
        self.right = child  
def preorder(self, node):  
  
    res=[]  
    if node:  
        res.append(node.data_value)  
        res = res + self.preorder(node.left)  
        res = res + self.preorder(node.right) return res
```

### Execution

```
# Root_Node  
print("Create Root Node")  
root = Node("Root_Node")  
#Tree_Left  
print("Create Tree_Left")  
tree_left = Node("Tree_Left")  
root.insert_left(tree_left)  
#Tree_Right  
print("Create Tree_Right")  
tree_right = Node("Tree_Right")  
root.insert_right(tree_right)  
#TreeLL  
print("Create TreeLL")  
treell = Node("TreeLL")  
tree_left.insert_left(treell)  
print("*****Preorder Traversal*****")  
print(root.preorder(root))
```

## Output

```
Create Root Node
Create Tree_Left
Create Tree_Right
Create TreeLL
*****Preorder Traversal*****
['Root_Node', 'Tree_Left', 'TreeLL', 'Tree_
Right']
>>>
```

In Order Traversal :First visit all nodes on the left then the root node and then all nodes on the right..

```
class Node(object):
    def __init__(self, data_value):
        self.data_value = data_value
        self.left = None
        self.right = None
    def insert_left(self, child):
        if self.left is None:
            self.left = child
        else:
            child.left = self.left
            self.left = child
    def insert_right(self, child):
        if self.right is None:
            self.right = child
        else:
            child.right = self.right
            self.right = child
    def inorder(self, node):
        res=[]
        if node:
            res = self.inorder(node.left)
            res.append(node.data_value)
            res = res + self.inorder(node.right)  return res
```

```
# Root_Node
print("Create Root Node")
root = Node("Root_Node")
#Tree_Left
print("Create Tree_Left")
tree_left = Node("Tree_Left")
root.insert_left(tree_left)
#Tree_Right
print("Create Tree_Right")
tree_right = Node("Tree_Right")
root.insert_right(tree_right)
#TreeLL
print("Create TreeLL")
treell = Node("TreeLL")
tree_left.insert_left(treell)
print("*****Inorder Traversal*****")
print(root.inorder(root))
```

Output

```
Create Root Node
Create Tree_Left
Create Tree_Right
Create TreeLL
*****Inorder Traversal*****
['TreeLL', 'Tree_Left', 'Root_Node', 'Tree_Right'] >>>
```

Post Order Traversal: Visit all nodes on the left , then visit all nodes on the right , then visit the root node.

## Code

```

class Node(object):
    def __init__(self, data_value):
        self.data_value = data_value
        self.left = None
        self.right = None
    def insert_left(self, child):
        if self.left is None:
            self.left = child
        else:
            child.left = self.left
            self.left = child
    def insert_right(self, child):
        if self.right is None:
            self.right = child
        else:
            child.right = self.right
            self.right = child
    def postorder(self, node):
        def postorder(self, node):
            res=[]
            if node:
                res = self.postorder(node.left)
                res = res + self.postorder(node.right)
                res.append(node.data_value)

            return res

```

## Execution

```

# Root_Node
print("Create Root Node")
root = Node("Root_Node")
#Tree_Left
print("Create Tree_Left")
tree_left = Node("Tree_Left")
root.insert_left(tree_left)
#Tree_Right
print("Create Tree_Right")
tree_right = Node("Tree_Right")

```

```

root.insert_right(tree_right)
#TreeLL
print("Create TreeLL")
treell = Node("TreeLL")
tree_left.insert_left(treell)
print("*****Postorder Traversal*****")
print(root.postorder(root))
OUTPUT
Create Root Node
Create Tree_Left
Create Tree_Right
Create TreeLL
*****Postorder Traversal*****
['TreeLL', 'Tree_Left', 'Tree_Right', 'Root_Node']

```

A binary heap is a binary tree. It is a complete tree, which means that all levels are completely filled except possibly the last level. It also means that the tree is balanced

As every new item is inserted next to the available space. A Binary heap can be stored in an array.

A binary heap can be of two types: Min Heap or Max Heap. In case of Min Heap binary heap, the root node is the minimum of all the nodes in the binary heap, all parent nodes are smaller than their children. On the other hand in case of Max Heap the root is the maximum among all the keys present in the heap and all nodes are bigger than their children.

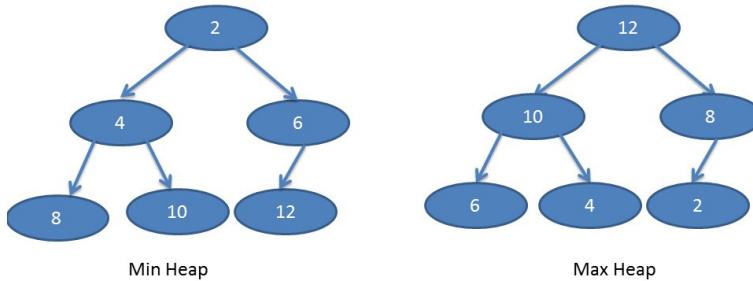


Figure 50

Heap has two important properties:

1. It is complete and is constructed from left to right across each row and the last row may not be completely full. Each row should be filled

up sequentially. So, the order of inserting values should be from left to right, row by row sequentially as shown in the following figure:

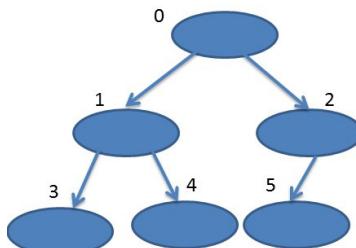


Figure 51

2. The parent must be larger (Max Heap)/smaller(Min Heap) than the children.

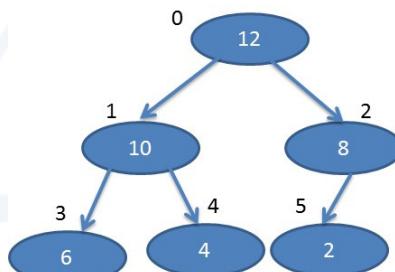


Figure 52

Look at the figure shown above, it is a case of Maximum heap because all parents are greater than their children.

The binary heap can be represented in an array as follows:

| 0  | 1  | 2 | 3 | 4 | 5 |
|----|----|---|---|---|---|
| 12 | 10 | 8 | 6 | 4 | 2 |

Figure 53

If you look at the array carefully you will realize that if a parent exists at location  $n$  then the left child is at  $2n+1$  and the right child is at  $2n+2$ .

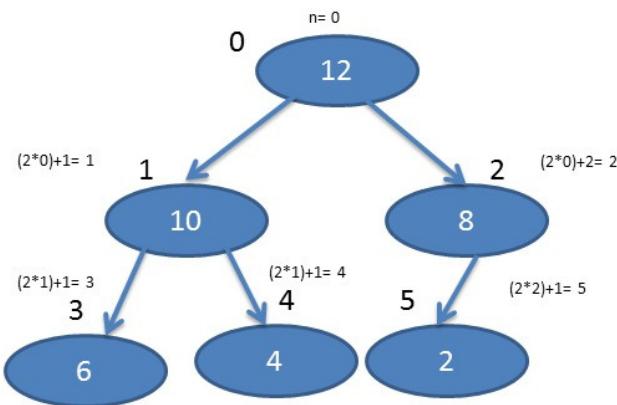


Figure 54

So, if we know the location of the parent child we can easily find out the location of the left and right child.

Now suppose we have to build up a Max Heap using following values:

20, 4, 90, 1, 125

Step 1:

Insert 20



Figure 55

Step 2:

Insert 4 -> Left to Right-> First element in second row

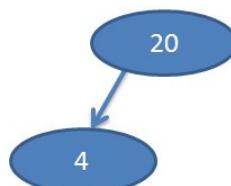


Figure 56

Since the parent is greater than the child, this is fine.

Step 3:

Insert 90 -> Left to Right -> Second element in second row

However when we insert 90 as the right child, it violates the rule of the max heap because the parent has a key value of 20. So, in order to resolve this problem it is better to swap places as shown in the following figure:

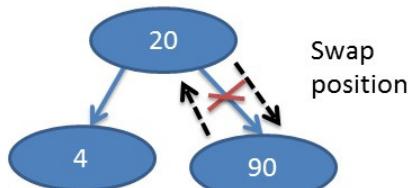


Figure 57

After swapping, the heap would look something like this:

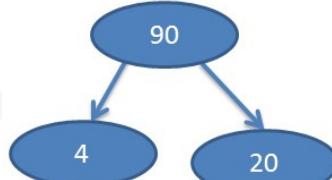


Figure 58

Step 4:

Insert 1 -> left to right -> first element third row

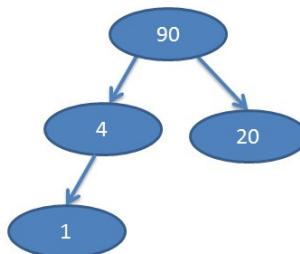


Figure 59

Since 1 is smaller than the parent node, this is fine.

Step 5:

Insert 125

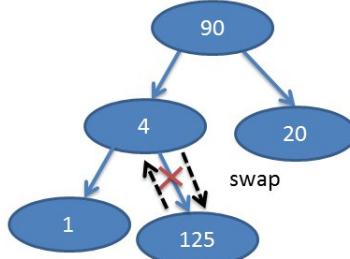


Figure 60

This violated the rule of Max Heap

Swap 4 and 125

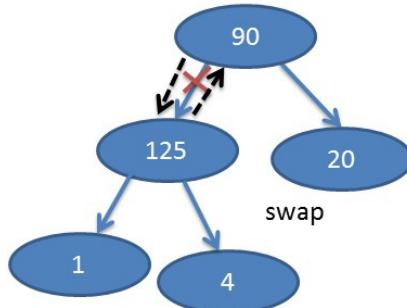


Figure 61

Not a heap still

Swap 125 and 90

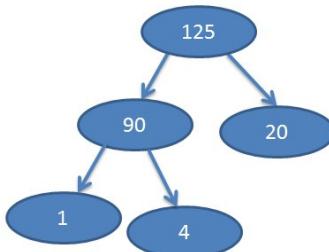


Figure 62

We now have a Max Heap.

**Question:** Write python code to implement Max Heap. How would you insert a value so that the root node has the maximum value and all parents are greater than their children.

*Answer :* Here we will write The code to implement Maxheap class will have two functions:

Push() : To insert value.

Float\_up() : To place the value where it belongs.

Step 1

Define the class

```
class MaxHeap:
```

Step 2

Define The Constructor

```
def __init__(self):
    self.heap = []
```

Step 3

Define the push() function

The push function does two jobs:

1. Appends the value at the end of the list. (We have seen earlier that the value has to be inserted first at the available location)
2. After appending the value, the push() function will call the float\_up(index) function. The push() function passes the index of the last element to the float\_up() function so that the float\_up() function can analyse the values and do the needful as described in the next step.

```
def push(self,value):
    self.heap.append(value)
    self.float_up(len(self.heap)-1)
```

Step 4

Define the float\_up function

1. The float\_up() function takes index value as parameter.

```
def float_up(self,index):
```

2. The push() function passes the index value of the last element in the heap. The first thing that the float function does is that it checks if the element is at index 0 or not. If yes, then it is the root node. Root node has no parents and since it's the last element of the heap it has no children either. So, we can return the value.

```
if index==0:  
    return
```

3. If the index is greater than 0 then we can proceed further. Look the following figure once again:

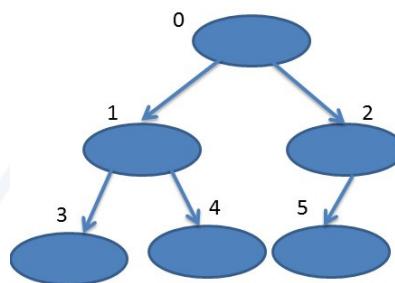


Figure 63

For programming, you need to understand few things:

The index 0 has two children at position 1 and 2. If I have an element at 1 then I can find the parent by calculating value of  $(1//2)$ . Similarly for element at index 2, the parent's index value can be found out by calculating value of  $(2//2 - 1)$ . So, we can say that if an element has an index value index and it is odd then it's parent's index value, `parent_of_index = index//2`. However, if index is even `parent_of_index` will be  $(index//2)-1$ . This becomes the outer frame for the `float_up` function. The right child has an even number as index and the left child has an odd number as index.

```

if index==0:
    return
else:
    if index%2==0:
        parent_of_index = (index//2)-1
        -----write code-----

    else:
        parent_of_index = index//2
        -----write code-----

```

4. Now, we compare the values of the child and the parent. If the child is greater than the parent, swap the elements.

```

def float_up(self,index):
    if index==0:
        return
    else:
        if index%2==0:
            parent_of_index = (index//2)-1
            if self.heap[index]> self.heap[parent_of_index]:
                self.swap(index, parent_of_index)
            else:
                parent_of_index = index//2
                if self.heap[index]> self.heap[parent_of_index]:
                    self.swap(index, parent_of_index)
                    self.float_up(parent_of_index)

```

## Step 5

Define the swap function

The `swap()` function helps in swapping the values of the parent and the child. the code for this is given as follows:

```
def swap(self, index1, index2):
    temp = self.heap[index1]
    self.heap[index1] = self.heap[index2]
    self.heap[index2] = temp
```

Code

```
class MaxHeap:
    def __init__(self):
        self.heap = []
    def push(self, value):
        self.heap.append(value)
        self.float_up(len(self.heap)-1)
    def float_up(self, index):
        if index==0:
            return
        else:
            if index%2==0:
                parent_of_index = (index//2)-1
            if self.heap[index]> self.heap[parent_of_index]:
                self.swap(index, parent_of_index)
            else:
                parent_of_index = index//2
            if self.heap[index]> self.heap[parent_of_index]:
                self.swap(index, parent_of_index)
                self.float_up(parent_of_index)

    def peek(self):
        print(self.heap[0])
    def pop(self):
        if len(self.heap)>=2:
            temp = self.heap[0]
            self.heap[0]=self.heap[len(self.heap)-1]
            self.heap[len(self.heap)-1]
            self.heap.pop()
            self.down_adj()
        elif len(self.heap)==1:
            self.heap.pop()
        else:
```

```

print("Nothing to pop")
def swap(self,index1, index2):
    temp = self.heap[index1]
    self.heap[index1] = self.heap[index2]
    self.heap[index2] = temp

```

### Execution

```

H = MaxHeap()
print("*****pushing      values*****")
print("pushing 165")
H.push(165)
print(H.heap)
print("pushing 60")
H.push(60)
print(H.heap)
print("pushing 179")
H.push(179)
print(H.heap)
print("pushing 400")
H.push(400)
print(H.heap)
print("pushing 6")
H.push(6)
print(H.heap)
print("pushing 275")
H.push(275)
print(H.heap)

```

### Output

```

*****pushing      values*****
pushing 165
[165]
pushing 60
[165, 60]
pushing 179
[179, 60, 165]
pushing 400
[400, 179, 165, 60]
pushing 6
[400, 179, 165, 60, 6]
pushing 275
[400, 179, 275, 60, 6, 165] >>>

```

**Question: Write code to find out the maximum value in the Max heap.**

*Answer:* It is easy to find the max value in the max heap as the maximum value is available at the root node which is index 0 of the heap.

```
def peek(self):
    print(self.heap[0])
```

**Question: Write the code to pop the maximum value from Max Heap.**

*Answer:* There are two steps involved:

1. Swap the root with the last element in the array and pop the value.
2. The root node now does have the maximum value. So, now we need to move downwards, comparing the parent with their left and right child to ensure that the children are smaller than the parent. If not we will have to swap places again.

Step 1

Define pop() function.

pop() The function which swaps the values of the root node with the last element in the list and pops the value. The function then calls the down\_adj() function which moves downwards and adjusts the values. The pop() function first checks the size of the heap. If the length of the heap is 1 then that means that it only contains one element that's the root and there is no need to swap further.

```
def pop(self):
    if len(self.heap)>2:
        temp = self.heap[0]
        self.heap[0]=self.heap[len(self.heap)-1]
        self.heap[len(self.heap)-1]
        self.heap.pop()
        print("heap after popping largest value =", self.heap)
        self.down_adj()
    elif len(self.heap)==1:
        self.heap.pop()
    else:
        print("Nothing to pop")
```

Step 2:

Define down\_adj() function

Set Index Value To 0.

*Index of left child = left\_child = index\*2+1*

*Index of right child = right\_child = index\*2+2*

Then we go through loop where we check the value of the parent with both left and right child. If the parent is smaller than the left child then we swap the value. We then compare the parent value with the right child, if it is less then we swap again.

This can be done as follows:

- Check if parent is less than left child:
  - O If yes, check if left child is less than the right child
  - O if yes, then swap the parent with the right child
  - O Change the value of index to value of right\_child for further assessment
  - If no the just swap the parent with the left child
  - Set the value of index to value of left\_child for further assessment
  - If the parent is not less than left child but only less than the right child then swap values with the right child
  - Change the value of index to right\_child

```
def down_adj(self):
    index = 0
    for i in range(len(self.heap)//2):
        left_child = index*2+1
        if left_child > len(self.heap):
            return
        print("left child = ", left_child)
        right_child = index*2+2
        if right_child > len(self.heap):
            return
        print("right child = ", right_child)
        if self.heap[index]<self.heap[left_child]:
            temp = self.heap[index]
            self.heap[index] = self.heap[left_child]
            self.heap[left_child] = temp
            index = left_child
        if self.heap[index]<self.heap[right_child]:
            temp = self.heap[index]
            self.heap[index] = self.heap[right_child]
            self.heap[right_child] = temp
            index = right_child
```

Code

```
class MaxHeap:  
    def __init__(self):  
        self.heap = []  
    def push(self,value):  
        self.heap.append(value)  
        self.float_up(len(self.heap)-1)  
    def float_up(self,index):  
        if index==0:  
            return  
        else:  
            if index%2==0:  
                parent_of_index = (index//2)-1  
            if self.heap[index]> self. heap[parent_of_index]:  
                temp = self.heap[parent_of_ index]  
                self.heap[parent_of_index] = self.heap[index]  
                self.heap[index] = temp  
            else:  
                parent_of_index = index//2  
            if self.heap[index]> self. heap[parent_of_index]:  
                temp = self.heap[parent_of_ index]  
                self.heap[parent_of_index] = self.heap[index]  
                self.heap[index] = temp  
                self.float_up(parent_of_index)  
  
    def peek(self):  
        print(self.heap[0])  
    def pop(self):  
        if len(self.heap)>=2:  
            temp = self.heap[0]  
            self.heap[0]=self.heap[len(self.heap)-1]  
            self.heap[len(self.heap)-1]  
            self.heap.pop()  
            self.down_adj()  
        elif len(self.heap)==1:  
            self.heap.pop()
```

```
else:  
    print("Nothing to pop")  
def swap(self,index1, index2):  
    temp = self.heap[index1]  
    self.heap[index1] = self.heap[index2]  
    self.heap[index2] = temp  
  
def down_adj(self):  
    index = 0  
    for i in range(len(self.heap)//2):  
        left_child = index*2+1  
        if left_child > len(self.heap)-1:  
            print(self.heap)  
            print("End Point")  
            print("Heap value after pop() = ",self.heap)  
            return  
  
        right_child = index*2+2  
        if right_child > len(self.heap)-1:  
            print("right child does not exist")  
        if self.heap[index]<self.heap[left_child]:  
            self.swap(index,left_child)  
        index = left_child  
        print("Heap value after pop() = ",self.heap)  
        return  
        if self.heap[index]<self.heap[left_child]:  
            if self.heap[left_child]<self.heap[right_child]:  
                self.swap(index,right_child)  
            index = right_child  
        else:  
            self.swap(index,left_child)  
        index = left_child  
        elif self.heap[index]<self.heap[right_child]:  
            self.swap(index,right_child)  
        index = right_child  
    else:  
        print("No change required" )  
        print("Heap value after pop() = ",self.heap)
```

## Execution

```
H = MaxHeap()  
print("*****pushing      values*****")  
H.push(165)  
print(H.heap)  
H.push(60)  
print(H.heap)  
H.push(179)  
print(H.heap)  
H.push(400)  
print(H.heap)  
H.push(6)  
print(H.heap)  
H.push(275)  
print(H.heap)  
print("*****popping      values*****")  
H.pop()  
H.pop()  
H.pop()  
H.pop()  
H.pop()  
H.pop()  
H.pop()
```

## Output

```
pushing values  
[165]  
[165, 60]  
[179, 60, 165]  
[400, 179, 165, 60]  
[400, 179, 165, 60, 6]  
[400, 179, 275, 60, 6, 165]  
*****popping values*****  
[275, 179, 165, 60, 6]  
End Point  
Heap value after pop() = [275, 179, 165, 60, 6] right  
child does not exist  
Heap value after pop() = [179, 60, 165, 6]  
Heap value after pop() = [165, 60, 6]  
right child does not exist  
Heap value after pop() = [60, 6]  
Heap value after pop() = [6]  
Nothing to pop  
>>>
```

**Question: Time complexity for max heap:****Answer:**

- Insert:  $O(\log n)$
- Get Max:  $O(1)$  because the maximum value is always the root at index 0
- Remove Max:  $O(\log n)$

The implementation of Min Heap is similar to Max Heap just that in this case the root has the lowest value and the value of parent is less than the left and right child.

**Code**

```
class MinHeap:
    def __init__(self):
        self.heap = []
    def push(self,value):
        self.heap.append(value)
        self.float_up(len(self.heap)-1)
    def float_up(self,index):
        if index==0:
            return
        else:
            if index%2==0:
                parent_of_index = (index//2)-1
            if self.heap[index]< self.heap[parent_of_index]:
                self.swap(index, parent_of_index)
            else:
                parent_of_index = index//2
            if self.heap[index]< self.heap[parent_of_index]:
                self.swap(index, parent_of_index)
            self.float_up(parent_of_index)

    def peek(self):
        print(self.heap[0])
    def pop(self):
        if len(self.heap)>=2:
            temp = self.heap[0]
```

```
        self.heap[0]=self.heap[len(self.heap)-1]
self.heap[len(self.heap)-1]
self.heap.pop()
self.down_adj()
elif len(self.heap)==1:
self.heap.pop()
else:
print("Nothing to pop")
def swap(self,index1, index2):
temp = self.heap[index1]
self.heap[index1] = self.heap[index2]
self.heap[index2] = temp
```

### Execution

```
H = MinHeap()
print("*****pushing      values*****")
print("pushing 165")
H.push(165)
print(H.heap)
print("pushing 60")
H.push(60)
print(H.heap)
print("pushing 179")
H.push(179)
print(H.heap)
print("pushing 400")
H.push(400)
print(H.heap)
print("pushing 6")
H.push(6)
print(H.heap)
print("pushing 275")
H.push(275)
print(H.heap)
```

## Output

```
*****pushing      values*****
pushing 165
[165]
pushing 60
[60, 165]
pushing 179
[60, 165, 179]
pushing 400
[60, 165, 179, 400]
pushing 6
[6, 60, 179, 400, 165]
pushing 275
[6, 60, 179, 400, 165, 275] >>>
```

### **Question: What are the applications of binary heap?**

*Answer:*

- Dijkstra Algorithm
- Prims Algorithm
- Priority Queue
- Can be used to solve problems such as:
  - O K'th Largest Element in an array
  - O Sort an almost sorted array
  - O Merge K Sorted Array

### **Question: What is a priority queue and how can it be implemented?**

*Answer:* Priority queue is like a queue but it is more advanced. It has methods same as a queue. However, the main difference is that it keeps the value of higher priority at front and the value of lowest priority at the back. Items are added from the rear and removed from the front. In priority queue elements are added in order. So, basically there is a priority associated with every element. Element of highest priority is removed first. Elements of equal priority are treated ass per their order in queue.

Binary heap is the most preferred way of implementing priority queue as they can retrieve the element of highest priority in  $O(1)$  time. Insert and delete can take  $O(\log n)$  time. Besides that since, the binary heap use lists or arrays, it is easy to locate elements and the processes involved are definitely quite cache friendly. Binary heaps do not demand extra space for pointers and are much easier to insert.

# Chapter 14

## Searching and Sorting

### Sequential Search

In the chapter based on Python operators we have seen that we can make use of membership operator ‘in’ to check if a value exists in a list or not.

```
>>> list1 = [1,2,3,4,5,6,7] >>> 3 in  
list1  
True  
>>> 8 in list1  
False  
>>>
```

This is nothing but searching for an element in a list. We are going to look at how elements can be searched and how process efficiency can be improved. We start by learning about sequential search.

The simplest way of searching for an element in a sequence is to check every element one by one. If the element is found then the search ends and the element is returned or else the search continues till the end of the sequence. This method of searching is known as linear search or sequential search. It follows a simple approach but it is quite inefficient way of searching for an element because we move from one element to the other right from beginning to end and if the element is not present in the sequence we would not know about it till we reach the last element.

Time analysis for sequential search:

- Σ Best scenario is that the element that we are looking for is the very first element in the list. In this case the time complexity will be  $O(1)$ .
- Σ The worst scenario would be if we traverse throughout the entire sequence and realize that the element that we are looking for, does not exist. In this case the time complexity will be  $O(n)$ .

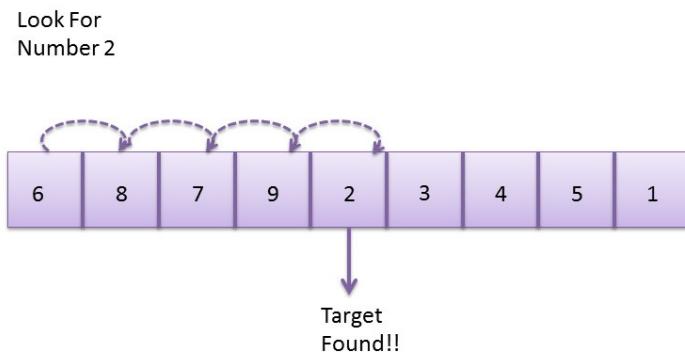


Figure 64

**Question:** Sequential search is also known as \_\_\_\_\_

**Answer:** Linear Search

**Question:** How are the elements reviewed in sequential search?

**Answer:** The elements are reviewed one at a time in sequential term.

**Question:** When does the sequential search end?

**Answer:** The sequential search ends when an element is found or when the end of the sequence is reached.

**Question:** Write code to implement sequential search.

**Answer:** The sequential search can be implemented as follows:

- The function takes two values: seq\_list which is the list and target\_num which is the number to search in the list.
- We set search\_flag = 0 if the target number is found in the list we will set the search\_flag to 1 else we let it be 0.
- We iterate through the list comparing each element in the list with the target\_num.
- If a match is found we print a message and update the search\_flag to 1. After the “for” loop if the search\_flag is still 0 then it means that the number was not found.

### Code

```
def sequential_search(seq_list, target_num):
    search_flag = 0
    for i in range(len(seq_list)):
        if seq_list[i] == target_num:
            print("Found the target number ", target_num,
                  " at index", i,".")
            search_flag = 1;
        if search_flag == 0:
            print("Target Number Does Not Exist. Search
                  Unsuccessful.")
```

### Execution

```
seq_list = [1,2,3,4,5,6,7,8,2,9,10,11,12,13,14,15, 16]
target_num = input("Please enter the target number
: ")
sequential_search(seq_list, int(target_num))
```

### Output 1

```
Please enter the target number : 5
Found the target number 5 at index 4 .
```

### Output 2

```
Please enter the target number : 2
Found the target number 2 at index 1 .
Found the target number 2 at index 8 .
```

### Output 3

```
Please enter the target number : 87
Target Number Does Not Exist. Search Unsuccessful.
```

**Question: How would you implement sequential search for an ordered list?**

**Answer:** When elements in a list are sorted, then many times there may not be the need to scan the entire list. The moment we reach an element that has a value greater than the target number, we know that we need not go any further.

**Step 1:** We define a function `sequential_search()` that takes two

arguments

- a list (`seq_list`) and the number that we are looking for (`target_num`).

```
def sequential_search(seq_list, target_num):
```

Step 2: The first thing we do is set define a flag (search\_flag) and set it to “False” or “0” value. The flag is set to “True” or “1” if the element is found. So, after traversing though the list if the search\_flag value is still “False” or “0”, we would know that the number that we are looking for does not exist in the list.

```
def sequential_search(seq_list, target_num):
    search_flag = 0
```

Step 3: Now, it’s time to check the elements one by one so, we define the for loop:

```
def sequential_search(seq_list, target_num):
    search_flag = 0
    for i in range(len(seq_list)):
```

Step 4: We now define how the elements are compared. Since it is an ordered list for every “i” in seq\_list we have to check if  $i > \text{target\_num}$ . If yes, then it means that there is no point moving further as it is an ordered list and we have reached an element that is greater than the number that we are looking for. However if  $\text{seq\_list}[i] == \text{target\_num}$  then, the search is successful and we can set the search\_flag to 1.

```
def sequential_search(seq_list, target_num):
    search_flag = 0
    for i in range(len(seq_list)):
        if seq_list[i] > target_num:
            print("search no further.")
            break;
        elif seq_list[i] == target_num:
            print("Found the target number ", target_num,
                  " at index", i, ".")
            search_flag = 1;
```

Step 5: After the for loop has been executed if the value of search\_flag is still 0 then a message stating that the target number was not found must be displayed.

### Code

```
def sequential_search(seq_list, target_num):
    search_flag = 0
    for i in range(len(seq_list)):
        if seq_list[i] > target_num:
            print("search no further.")
            break;
        elif seq_list[i] == target_num:
            print("Found the target number ", target_num,
                  " at index", i,".")
            search_flag = 1;

    if search_flag == 0:
        print("Target Number Does Not Exist. Search
Unsuccessful.")
```

### Execution

```
seq_list = [1,2,3,4,5,6,7,8,2,9,10,11,12,13,14,15, 16]
target_num = input("Please enter the target number
: ")
sequential_search(seq_list, int(target_num))
```

### Output 1

```
Please enter the target number : 2
Found the target number 2 at index 1 .
Found the target number 2 at index 2 .
search no further.
>>>
```

### Output 2

```
Please enter the target number : 8
Found the target number 8 at index 8 .
search no further.
>>>
```

### Output 3

```
Please enter the target number : 89
Target Number Does Not Exist. Search Unsuccessful.
>>>
```

### Binary Search

Binary search is used to locate a target value from a sorted list. The search begins from the center of the sequence. The element present at the centre is not equal to the target number it is compared with the target number. If the target number is greater than the element at the center then it means that we need to search for the number in the right half of the list and the left half need not be touched. Similarly, if the target number is less than the element present in the center then we will have to direct our search efforts towards the left. This process is repeated till the search is completed. The beauty of binary search is that in every search operation the sequence is cut into half and focus is shifted to only that half that has chances of having the value.

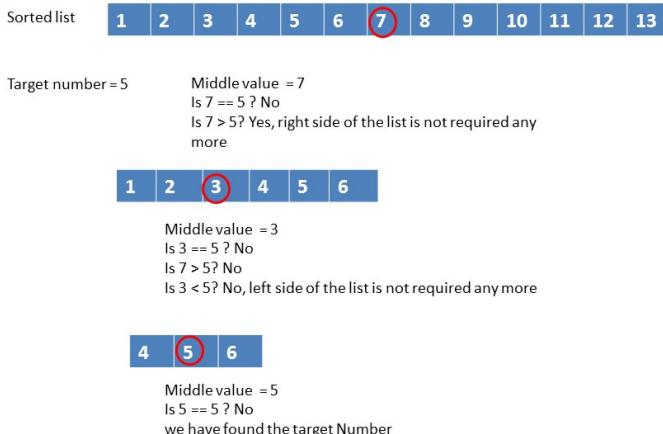


Figure 65

### Question: Write a code to implement binary search function.

Answer: The binary search can be implemented in the following manner:

Step 1 Define the binary\_search function. It takes 4 parameters:

- sorted\_list : the input list that is in sorted form
- target\_num : the number that we are looking for

- starting\_point: the place from where we want to start searching, default value = 0

- end\_point: The end point for search, default value = None

Note that the list will be split into half in every step so the starting and ending point may change in every search operation.

```
def binary_search(sorted_list, target_num, start_point=0, end_point=None):
```

Step 2: Do the following:

- Set the search\_flag to “False”
- If the end\_point is not provided, it would have the default value of “None”, set it to the length of the input list.

```
def binary_search(sorted_list, target_num,
                  start_point=0, end_point=None):
    search_flag = False
    if end_point == None:
        end_point = len(sorted_list)-1
```

Step 3: Check, the start\_point should be less than the end point. If that is true, do the following:

- Get midpoint index value: mid\_point = (end\_point+start\_point)//2
- Check the value of the element at mid\_point. Is it equal to the target\_num?
  - o If sorted\_list[mid\_point] == target\_num?
  - o Set search\_flag to True
  - If not check if value at mid\_point is greater than target\_num :
    - o sorted\_list[mid\_point] > target\_num
      - o If yes, then we can discard the right side of the list now we can repeat search from beginning to mid\_point-1 value. Set end point to mid\_point - 1. The starting point can remain the same(0).

- o The function should now call itself with:

sorted\_list : same as before  
 target\_num : same as before  
 starting\_point: same as before  
 end\_point: mid\_point – 1

- If not check if value at mid\_point is lesser than target\_num :
  - $\text{sorted\_list}[\text{mid\_point}] < \text{target\_num}$
  - If yes, then left side of the list is not required. We can repeat search from mid\_point+1 to end of the list. Set starting point to mid\_point+1. The ending\_point can remain the same.
  - The function should now call itself with:
    - sorted\_list : same as before
    - target\_num : same as before
    - starting\_point: mid\_point+1
    - end\_point: same as before
- If at the end of this procedure the search\_flag is still set to “False”, then it means that the value does not exist in the list.

Code

```
def binary_search(sorted_list, target_num, start_
point=0, end_point=None):
    search_flag = False
    if end_point == None:
        end_point = len(sorted_list)-1
    if start_point < end_point:
        mid_point = (end_point+start_point)//2
        if sorted_list[mid_point] == target_num:
            search_flag = True
            print(target_num," Exists in the list at",
",sorted_list.index(target_num))")
        elif sorted_list[mid_point] > target_num:
            end_point = mid_point-1
            binary_search(sorted_list, target_num,start_point,
            end_point)
        elif sorted_list[mid_point] < target_num:
            start_point = mid_point+1
            binary_search(sorted_list, target_num, start_point,
            end_point)
    elif not search_flag:
        print(target_num," Value does not exist")
```

## Execution

```
sorted_list=[1,2,3,4,5,6,7,8,9,10,11,12,13]
binary_search(sorted_list,14)
binary_search(sorted_list,0)
binary_search(sorted_list,5)
```

## Output

```
14 Value does not exist
0 Value does not exist
5 Exists in the list at 4
```

## Hash Tables

Hash Tables are data structures where a hash function is used to generate the index or address value for a data element. It is used to implement an associative array which can map keys to values. The benefit of this is that it allows us to access data faster as the index value behaves as a key for data value. Hash tables store data in key-value pairs but the data is generated using the hash function. In Python the Hash Tables are nothing but Dictionary data type. Keys in the dictionary are generated using a hash function and the order of data elements in Dictionary is not fixed. We have already learnt about various functions that can be used to access a dictionary object but what we actually aim at learning here is how hash tables are actually implemented.

We know that by binary search trees we can achieve time complexity of  $O(\log n)$  for various operations. The question that arises here is that can search operations be made faster? Is it possible to reach time complexity of  $O(1)??$  This is precisely why hash tables came into existence. Like in a list or an array if the index is known, the time complexity for search operation can become  $O(1)$ . Similarly, if data is stored in key value pairs, the result can be retrieved faster. So, we have keys and we have slots available where the values can be placed. If we are able to establish a relationship between the slots and the key it would be easier for to

retrieve  
the value at a fast rate. Look at the following figure:

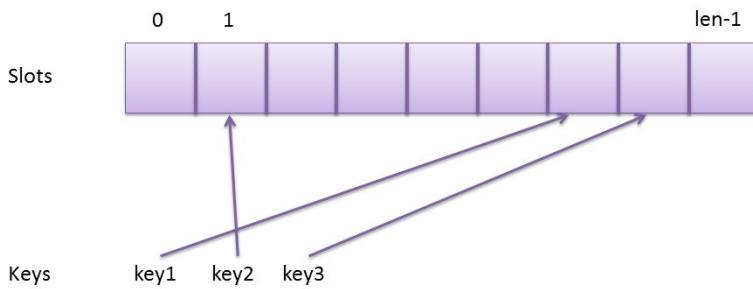


Figure 66

The key value is not always a non negative integer, it can be a string also, whereas the array has index starting from 0 to length\_of\_array -1. So there is a need to do prehashing in order to match the string keys to indexes. So, for every key there is a need to find an index in array where the corresponding value can be placed. In order to do this we will have to create a hash() function that can map a key of any type to a random array index.

During this process there are chances of collision. Collision is when we map two keys to the same index as shown in the following figure:

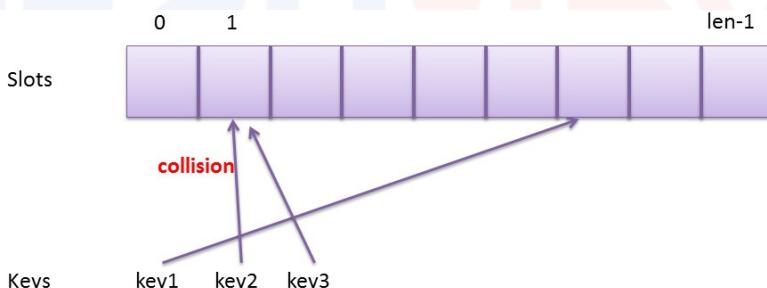


Figure 67

To resolve collision we can use chaining. Chaining is when values are stored in the same slot with the help of a linked list as shown in the following figure:

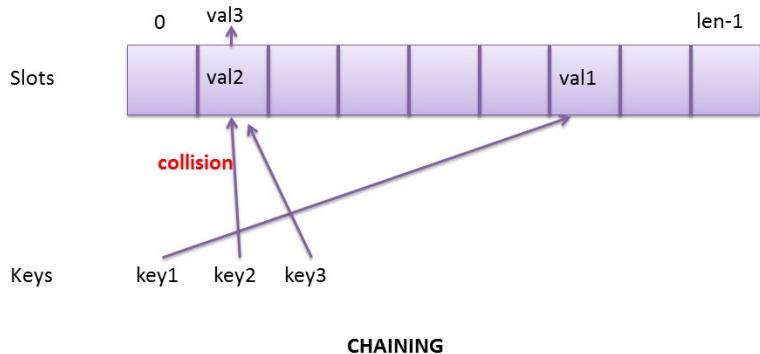


Figure 68

However, there can be cases of more than one collision for the same spot and considering the worst case scenario where there is a need to insert all values as elements of a linked list it can be a tough situation that will have a severe impact on the time complexity. Worst case scenario will be if we land up placing all values as linked list element at the same index.

To avoid this scenario we can consider the process of open addressing. Open addressing is a process of creating a new address. Consider a case where if there is a collision we increment the index by 1 and place the value there as shown below, there is collision while placing val3, as val2 already exists at index 1. So, the index value is incremented by 1 ( $1+1=2$ ) and val3 is placed at index 2.

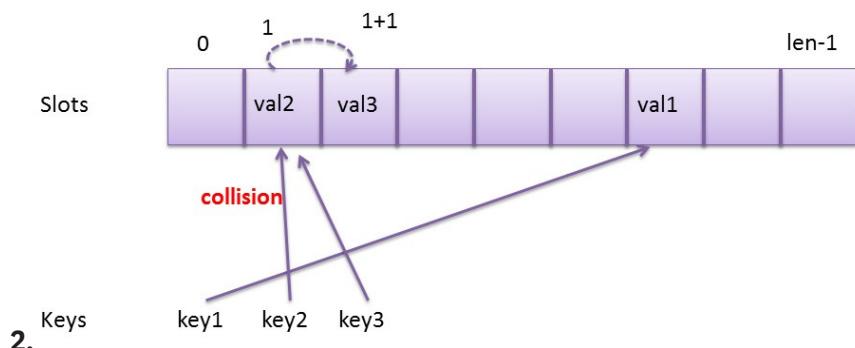


Figure 69

Had there been any other value at index 2 then the index would have incremented again and val3 could be placed at index 3. Which means that this process of incrementing index is continued till an empty slot is spotted. This is called Linear probing. Quadratic probing on the other hand increments by two times the index value. So, the search for empty slot is done at a distance of 1,2,4,8, and so on. *Rehashing* is the process of hashing the result obtained again to find an empty slot.

The purpose of the hash function is to calculate an index from which the right value can be found therefore its job would be:

1. To distribute the keys uniformly in the array.
2. If n is the number of keys and m is the size of an array, the  $\text{hash}() = n \% m$  (*modulo operator*) in case we use integers as keys.
  - a. Prefer to use prime numbers both for array and the hash function for uniform distribution
  - b. For string keys you can calculate the ascii value of each character and add them up and make modulo operator on them

In many scenarios, hash tables prove to be more efficient than the search trees and are often used in caches, databases and sets.

Important points:

1. You can avoid clustering by using prime numbers.
2. Number of entries divided by the size of the array is called load factor.
3. If the load factor increases the number of collisions will increase. This will reduce the performance of the hash table.
4. Resize the table when load factor exceeds the given threshold.

However, this would be an expensive option as the hashes of the values entered will change whenever resizing is done and this can take  $O(n)$  to complete. Hence dynamic size array may be inappropriate for real time scenarios.

#### **Question: What does the hash function do?**

**Answer:** The purpose of a hash function is to map the values or entries into the slots that are available in a hash table. So, for every entry the hash function will compute an integer value that would be in the range of 0 to  $m-1$  where m is the length of the array.

**Question:** What is a remainder hash function? What are the drawbacks? Write the code to implement remainder hash function.

**Answer:** The remainder hash function calculates the index value by taking one item at a time from collection. It is then divided by the size of the array and the remainder is returned.

$$h(item) = item \% m, \text{ where } m = \text{size of the array}$$

Let's consider the following array:

[18, 12, 45, 34, 89, 4]

The above array is of size 8.

| Item | Calculation= item%m | Result |
|------|---------------------|--------|
| 18   | 18%8                | 2      |
| 12   | 12%8                | 4      |
| 45   | 45%8                | 5      |
| 34   | 34%8                | 2      |
| 89   | 89%8                | 1      |
| 4    | 4%8                 | 4      |

Drawback: You can see here that 18 and 34 have same hash value of 2 and 12 and 4 have the same hash value of 4. This is a case of collision as a result when you execute the program, values 18 and 12 are replaced by 34 and 4 and you will not find these values in the hash table.

Let's have a look at the implementation:

Step1: Define the hash function that takes a list and size of array as input.  
**def hash(list\_items, size):**

```
def hash(list_items, size):
```

StepDo the following:

- Create an empty list.
- Now populate this key from the numbers 0 to size mention. This example takes a list of 8 elements so we are creating a list [0, 1, 2, 3, 4, 5, 6, 7].
- Convert this list to dict using fromkeys(). We should get a dictionary object of form {0: None, 1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None}. This value is assigned to hash\_table.

```
def hash(list_items, size):
    temp_list = []
    for i in range(size):
        temp_list.append(i)
    hash_table = dict.fromkeys(temp_list)
```

### Step3:

- Now iterate through the list.
- Calculate the index for every item by calculating  $item \% size$ .
- For the key value in the `hash_table = index`, insert the item.

### Code

```
def hash(list_items, size):
    temp_list = []
    for i in range(size):
        temp_list.append(i)
    hash_table = dict.fromkeys(temp_list)
    for item in list_items:
        i = item % size
        hash_table[i] = item
    print("value of hash table is : ", hash_table)
```

### Execution

```
list_items = [18, 12, 45, 34, 89, 4]
hash(list_items, 8)
```

### Output

```
value of hash table is : {0: None, 1: 89, 2: 34, 3: None,
4: 4, 5: 45, 6: None, 7: None}
>>>
```

### Question: What is folding hash function?

**Answer:** Folding hash function is a technique used to avoid collisions while hashing. The items are divided into equal-size pieces, they are added together and then the slot value is calculated using the same hash function ( $item \% size$ ).

Suppose, we have a phone list as shown below:

```
phone_list = [4567774321, 4567775514, 9851742433, 4368884732]
```

We convert every number to string, then each string is converted to a list and then each list is appended to another list and we get the following result:

```
[['4', '5', '6', '7', '7', '4', '3', '2', '1'], ['4', '5', '6', '7', '7', '7', '5', '5', '1', '4'], ['9', '8', '5', '1', '7', '4', '2', '4', '3', '3'], ['4', '3', '6', '8', '8', '8', '4', '7', '3', '2']]
```

Now from this new list, we take one list item one by one, for every item we concatenate two characters convert them to integer and then concatenate next to characters convert them to integer and add the two values and continue this till we have added all elements. The calculation will be something like this:

['4', '5', '6', '7', '7', '4', '3', '2', '1']

1. items = 4 5

string val = 45

integer value = 45

hash value= 45

2. items = 6 7

string val = 67

integer value = 67

hash value= 45+67 =112

3. items = 7 7

string val = 77

integer value = 77

hash value= 112+77 = 189

4. items = 4 3

string val = 43

integer value = 43

hash value= 189+43 = 232

5. items = 2 1

string val = 21

integer value = 21

hash value= 232+21 = 253

Similarly,

['4', '5', '6', '7', '7', '5', '5', '1', '4'] will have hash value of 511.

['9', '8', '5', '1', '7', '4', '2', '4', '3', '3'] will have hash value of 791.

## 256 - Python Interview Questions

[‘4’, ‘3’, ‘6’, ‘8’, ‘8’, ‘8’, ‘4’, ‘7’, ‘3’, ‘2’] will have a hash value of 1069. We now call the hash function for [253, 511, 791, 1069] for size 11.

| Item | Calculation= item%m | Result |
|------|---------------------|--------|
| 253  | 253%11              | 0      |
| 511  | 511%11              | 5      |
| 791  | 791%11              | 10     |
| 1069 | 1069%11             | 2      |

So, the result we get is:

{0: 253, 1: None, 2: 1069, 3: None, 4: None, 5: 511, 6: None, 7: None, 8: None, 9: None, 10: 791}

### Question: Write the code to implement coding hash function.

Answer:

Let's look at the execution statements for this program:

```
phone_list = [4567774321, 4567775514, 9851742433,  
4368884732]  
str_phone_values = convert_to_string(phone_list)  
folded_value = folding_hash(str_phone_values)  
folding_hash_table = hash(folded_value,11)  
print(folding_hash_table)
```

1. A list of phone numbers is defined: `phone_list = [4567774321, 4567775514, 9851742433, 4368884732]`
2. The next statement "`str_phone_values = convert_to_string(phone_list)`" calls a function `convert_to_string()` and passes the `phone_list` as argument. The function in turn returns a list of lists. The function takes one phone number at a time converts it to a list and adds to new list. So, we get the output as: [[‘4’, ‘5’, ‘6’, ‘7’, ‘7’, ‘7’, ‘4’, ‘3’, ‘2’, ‘1’], [‘4’, ‘5’, ‘6’, ‘7’, ‘7’, ‘5’, ‘5’, ‘1’, ‘4’], [‘9’, ‘8’, ‘5’, ‘1’, ‘7’, ‘4’, ‘2’, ‘4’, ‘3’, ‘3’], [‘4’, ‘3’, ‘6’, ‘8’, ‘8’, ‘8’, ‘4’, ‘7’, ‘3’, ‘2’]]. The following steps are involved in this function:
  - a. Define two lists a `phone_list[]`
  - b. For elements in `phone_list`, take every element i.e. the phone number one by one and:
    - i. convert the phone number to string: `temp_string = str(i)`
    - ii. Convert each string to list: `temp_list = list(temp_string)`
    - iii. Append the list obtained to the `phone_list` defined in previous step.

- iv. Return the phone\_list and assign values to str\_phone\_values

```
def convert_to_string(input_list):
    phone_list=[]
    for i in input_list:
        temp_string = str(i)
        temp_list = list(temp_string)
        phone_list.append(temp_list)
    return phone_list
```

3. The list str\_phone\_values is passed on to folding\_hash(). This method takes a list as input.
  - a. It will take each phone\_list element which is also a list.
  - b. Take one list item one by one.
    - c. For every item concatenate first two characters convert them to integer and then concatenate next to characters convert them to integer and add the two values.
    - d. Pop the first two elements from the list.
    - e. Repeat c and d till we have added all elements.
    - f. The function returns a list of hash values.

```
def folding_hash(input_list):
    hash_final = []
    while len(input_list) > 0:
        hash_val = 0
        for element in input_list:
            while len(element) > 1:
                string1 = element[0]
                string2 = element[1]
                str_combine = string1 + string2
                int_combine = int(str_combine)
                hash_val += int_combine
                element.pop(0)
                element.pop(0)
            if len(element) > 0:
                hash_val += element[0]
            else:
                pass
            hash_final.append(hash_val)
        return hash_final
```

4. Call hash function for size 11. The code for the hash function is same.

```
def hash(list_items, size):
    temp_list = []
    for i in range(size):
        temp_list.append(i)
    hash_table = dict.fromkeys(temp_list)  for
    item in list_items:
        i = item%size
        hash_table[i] = item
    return hash_table
```

Code

```
def hash(list_items, size):
    temp_list = []
    for i in range(size):
        temp_list.append(i)
    hash_table = dict.fromkeys(temp_list)
    for item in list_items:
        i = item%size
        hash_table[i] = item
    return hash_table
def convert_to_string(input_list):
    phone_list=[]
    for i in input_list:
        temp_string = str(i)
        temp_list = list(temp_string)
        phone_list.append(temp_list)
    return phone_list
def folding_hash(input_list):
    hash_final = []
    while len(input_list) > 0:
        hash_val = 0
        for element in input_list:
            while len(element) > 1:
                string1 = element[0]
                string2 = element[1]
                str_combine = string1 + string2  int_combine =
                int(str_combine)
                hash_val += int_combine
                element.pop(0)
                element.pop(0)
            if len(element) > 0:
                hash_val += element[0]
            else:
                pass
        hash_final.append(hash_val)
    return hash_final
```

### Execution

```
phone_list = [4567774321, 4567775514, 9851742433,
4368884732]
str_phone_values = convert_to_string(phone_list)
folded_value = folding_hash(str_phone_values)
folding_hash_table = hash(folded_value,11)
print(folding_hash_table)
```

### Output

```
{0: 253, 1: None, 2: 1069, 3: None, 4: None, 5:
511, 6: None, 7: None, 8: None, 9: None, 10: 791}
```

In order to store phone numbers at the index, we slightly change the hash() function;

1. The hash() function will take one more parameter : phone\_list
2. After calculating the index the corresponding element from the phone\_list is saved instead of the folded\_value.

```
def hash(list_items,phone_list, size):
temp_list = []
for i in range(size):
temp_list.append(i)
hash_table = dict.fromkeys(temp_list)
for i in range(len(list_items)):
hash_index = list_items[i] % size
hash_table[hash_index] = phone_list[i] return
hash_table
```

### Execution

```
phone_list = [4567774321, 4567775514, 9851742433,
4368884732]
str_phone_values = convert_to_string(phone_list)
folded_value = folding_hash(str_phone_values)
folding_hash_table = hash(folded_value,phone_
list,11)
print(folding_hash_table)
```

## Output

```
{0: 4567774321, 1: None, 2: 4368884732, 3: None,
4: None, 5: 4567775514, 6: None, 7: None, 8: None,
9: None, 10: 9851742433}
```

## Bubble sort

Bubble sort is also known as sinking sort or comparison sort. In bubble sort each element is compared with the adjacent element and the elements are swapped if they are found in wrong order. However this is a time consuming algorithm. It is simple but quite inefficient.

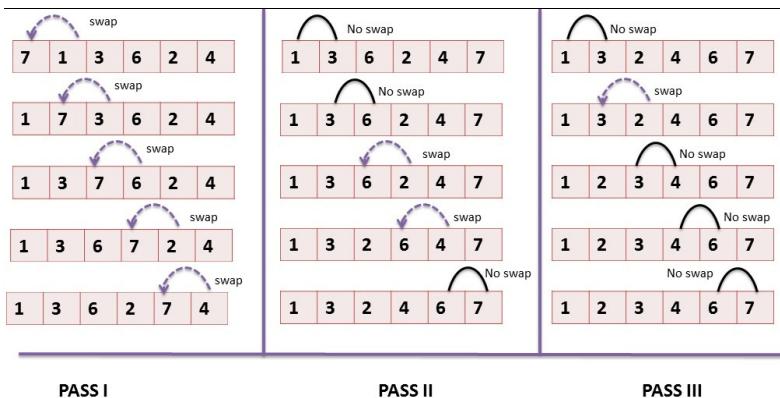


Figure 70

### Question: How will you implement bubble sort in Python?

**Answer:** The code for a bubble sort algorithm is very simple.

Step 1: Define the function for bubble sort. It would take the list that needs to be sorted as input.

```
def bubble_sort(input_list):
```

Step 2:

1. Set a loop for  $i$  in range  $\text{len}(\text{input\_list})$ 
  - a. Inside this for loop set another loop for  $j$  in range  $\text{len}(\text{input\_list})-i-1$ .
    - b. For every  $i$ , in the nested loop value at index  $j$  is compared with value at index  $j+1$ . If the value at index  $j+1$  is smaller than the value at index  $j$  then the values are swapped.
    - c. After the for loop is over print the sorted list.

## Code

```
def bubble_sort(input_list):
    for i in range(len(input_list)):
        for j in range(len(input_list)-i-1):
            if input_list[j]>input_list[j+1]:      temp =
input_list[j]
                input_list[j]=input_list[j+1]      input_list[j+1]=
temp
            print(input_list)
```

## Execution

```
x = [7,1,3,6,2,4]
print("Executing Bubble sort for ",x)
bubble_sort(x)

y = [23,67,12,3,45,87,98,34]
print("Executing Bubble sort for ",y)
bubble_sort(y)
```

## Output

```
Executing Bubble sort for [7, 1, 3, 6, 2, 4]
[1, 2, 3, 4, 6, 7]
Executing Bubble sort for [23, 67, 12, 3, 45, 87, 98, 34]
[3, 12, 23, 34, 45, 67, 87, 98]
```

**Question: Write code to implement selection sort.***Answer:*

Step 1: Define the function for `selection_sort` It would take the list that needs to be sorted as input.

```
def selection_sort(input_list):
```

Step 2:

1. Set a loop for `i` in `range(len(input_list))`
  - a. Inside this for loop set another loop for `j` in `range(i+1, len(input_list)-i-1)`
  - b. For every `i`, in the nested loop value at index `j` is compared with value at index `i`. If the value at index `j` is smaller than the value at index `i` then the values are swapped.

c. After the for loop is over print the sorted list.

Code

```
def selection_sort(input_list):
    for i in range(len(input_list)-1):
        for j in range(i+1, len(input_list)):
            if input_list[j] < input_list[i]:      temp = 
input_list[j]
            input_list[j] = input_list[i]  input_list[i] = temp
    print(input_list)
```

Execution

```
selection_sort([15,10,3,19,80,85])
```

Output

```
[3, 10, 15, 19, 80, 85]
```

Insertion Sort

In insertionsort each element at position x is compared with elements located at position x-1 to position 0. If the element is found to be less than any of the values that it is compared to then the values are swapped. This process is repeated till the last element has been compared.

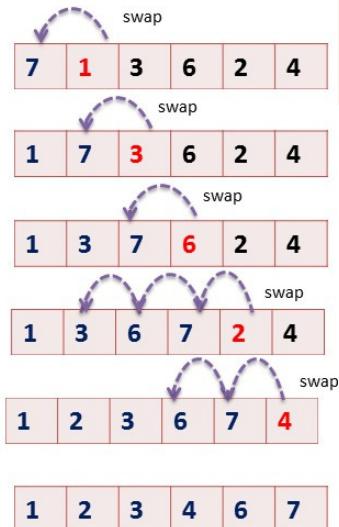


Figure 71

**Question: Write code to implement insertion sort.**

*Answer:* It is very easy to implement insertion sort:

Consider a list [7,1,3,6,2,4]

Let  $\text{index}_i = i$

$\text{index}_j = \text{index}_i + 1$

| $\text{index}_i$ | $\text{index}_j$  | $\text{val}[i] < \text{val}[j]$ | Swap | Change in index value                                                                   |
|------------------|-------------------|---------------------------------|------|-----------------------------------------------------------------------------------------|
| 0                | 1                 | 1<7 yes                         | yes  | $\text{index}_i = \text{index}_i - 1 = -1$<br>$\text{index}_j = \text{index}_j - 1 = 0$ |
| Iteration 2      | List: 1,7,3,6,2,4 |                                 |      |                                                                                         |
| 1                | 2                 | 3<7 yes                         | yes  | $\text{index}_i = \text{index}_i - 1 = 0$<br>$\text{index}_j = \text{index}_j - 1 = 1$  |
|                  | List: 1,3,7,6,2,4 |                                 |      |                                                                                         |
| 0                | 1                 | 3<1 no                          | no   | $\text{index}_i = \text{index}_i - 1 = -1$                                              |
| Iteration 3      | List: 1,3,7,6,2,4 |                                 |      |                                                                                         |
| 2                | 3                 | 6<7 yes                         | yes  | $\text{index}_i = \text{index}_i - 1 = 1$<br>$\text{index}_j = \text{index}_j - 1 = 0$  |
|                  | List: 1,3,6,7,2,4 |                                 |      |                                                                                         |
| 1                | 2                 | 7<3 n                           | n    | $\text{index}_i = \text{index}_i - 1 = 0$                                               |
| 0                | 2                 | 7<1 o                           | o    | $\text{index}_i = \text{index}_i - 1 = -1$                                              |
| Iteration 4      | List: 1,3,6,7,2,4 |                                 | n    |                                                                                         |
| 3                | 4                 | 2<7 yes                         | yes  | $\text{index}_i = \text{index}_i - 1 = 2$<br>$\text{index}_j = \text{index}_j - 1 = 1$  |
|                  | List: 1,3,6,2,7,4 |                                 |      |                                                                                         |
| 2                | 3                 | 2<6 yes                         | yes  | $\text{index}_i = \text{index}_i - 1 = 1$<br>$\text{index}_j = \text{index}_j - 1 = 2$  |
|                  | List: 1,3,2,6,7,4 |                                 |      |                                                                                         |
| 1                | 2                 | 2<3 yes                         | yes  | $\text{index}_i = \text{index}_i - 1 = 0$<br>$\text{index}_j = \text{index}_j - 1 = 1$  |
|                  | List: 1,2,3,6,7,4 |                                 |      |                                                                                         |
| 0                | 1                 | 2<1 no                          | no   | $\text{index}_i = \text{index}_i - 1 = -1$                                              |
| Iteration 5      | List: 1,2,3,6,7,4 |                                 |      |                                                                                         |

## 264 - Python Interview Questions

|   |                   |          |     |                                             |
|---|-------------------|----------|-----|---------------------------------------------|
| 4 | 5                 | 4<7 yes  | yes | indexi= indexi-1= 3<br>indexj = indexj-1 =4 |
|   | List: 1,2,3,6,4,7 |          |     |                                             |
| 3 | 4                 | 4<6 yes  | yes | indexi= indexi-1= 2<br>indexj = indexj-1 =1 |
|   | List: 1,2,3,4,6,7 |          |     |                                             |
| 2 | 3                 | 4<3 no   | No  | indexi= indexi-1=1                          |
| 1 | 3                 | 4<2 no   | No  | indexi= indexi-1=0                          |
| 0 | 3                 | 4 < 1 no | No  | indexi= indexi-1=-1                         |

Step 1: Define the `insert_sort()` function. It will take `input_list` as input.

```
def insertion_sort(input_list):
```

Step 2: for i in range(`input_list`-1), set `indexi = i`, `indexj = i+1`

```
for i in range(len(input_list)-1):  
    indexi = i  
    indexj = i+1
```

Step 3: Set while loop, condition `indexi>=0`

- if `input_list[indexi]>input_list[indexj]`
  - swap values of `input_list[indexi]` and `input_list[indexj]`
  - set `indexi = indexi -1`
  - set `indexj = indexj -1`
- else
  - set `indexi = indexi -1`

```
while indexi >= 0:  
    if input_list[indexi]>input_list[indexj]:  
        print("swapping")  
        temp = input_list[indexi]  
        input_list[indexi] = input_list[indexj]  
        input_list[indexj] = temp  
        indexi = indexi - 1  
        indexj = indexj - 1  
    else:  
        indexi = indexi - 1
```

Step 4: Print updated list

### Code

```
def insertion_sort(input_list):
    for i in range(len(input_list)-1):
        indexi = i
        indexj = i+1
        print("indexi = ", indexi)
        print("indexj = ", indexj)
        while indexi >= 0:
            if input_list[indexi]>input_list[indexj]:
                print("swapping")
                temp = input_list[indexi]
                input_list[indexi] = input_list[indexj]
                input_list[indexj] = temp
                indexi = indexi - 1
                indexj = indexj - 1
            else:
                indexi = indexi - 1
        print("list update:",input_list)
    print("final list = ", input_list)
```

### Execution

```
insertion_sort([9,5,4,6,7,8,2])
```

### Output

```
[7, 1, 3, 6, 2, 4]
indexi = 0
indexj = 1
swapping
list update: [1, 7, 3, 6, 2, 4] indexi
= 1
indexj = 2
swapping
list update: [1, 3, 7, 6, 2, 4] indexi
= 2
indexj = 3
swapping
list update: [1, 3, 6, 7, 2, 4] indexi
= 3
```

```

indexj = 4
swapping
swapping
swapping
list update: [1, 2, 3, 6, 7, 4] indexi
= 4
indexj = 5
swapping
swapping
list update: [1, 2, 3, 4, 6, 7] final
list = [1, 2, 3, 4, 6, 7] >>>

```

## Shell Sort

Σ Shell sort is a very efficient sorting algorithm.

Based on insertion sort.

Σ Implements insertion sort on widely spread elements at first and then in each step the space or interval is narrowed down.

Σ Great for medium size data set.

Worst case time complexity:  $O(n^2)$

Worst case space complexity :  $O(n)$

Σ I Consider a list: [10, 30, 11, 4, 36, 31, 15, 1]

Size of the list, n = 8

Divide  $n/2 = 4$ , let this value be named k

Σ Consider every kth(in this case 4th) element and sort them in right order.  
 Σ

|        | List | 10                                  | 30 | 11 | 4 | 36 | 31 | 15 | 1 |
|--------|------|-------------------------------------|----|----|---|----|----|----|---|
| n      | 8    |                                     |    |    |   |    |    |    |   |
| pass 1 | k    | $n/2 = 4$                           |    |    |   |    |    |    |   |
|        |      | 10                                  | 30 | 11 | 4 | 36 | 31 | 15 | 1 |
| step 1 |      | compare 10 and 36, no swap required |    |    |   |    |    |    |   |
|        |      | 10                                  | 30 | 11 | 4 | 36 | 31 | 15 | 1 |
| step 2 |      | compare 30 and 31, no swap required |    |    |   |    |    |    |   |
|        |      | 10                                  | 30 | 11 | 4 | 36 | 31 | 15 | 1 |
| step 3 |      | compare 11 and 15, no swap required |    |    |   |    |    |    |   |
|        |      | 10                                  | 30 | 11 | 4 | 36 | 31 | 15 | 1 |
| step 4 |      | compare 4 and 1, swap values        |    |    |   |    |    |    |   |
|        |      | 10                                  | 30 | 11 | 1 | 36 | 31 | 15 | 4 |

Figure 72

II do the following:

$$K = k/2 = 4/2 = 2$$

Consider every  $k$ th element and sort the order.

| <b><math>k = 4</math></b> |                       | List                 | 10 | 30 | 11 | 1 | 36 | 31 | 15 | 4  |
|---------------------------|-----------------------|----------------------|----|----|----|---|----|----|----|----|
| pass 2                    | <b><math>k</math></b> | $k/2 = 2$            |    |    |    |   |    |    |    |    |
|                           |                       |                      | 10 | 30 | 11 | 1 | 36 | 31 | 15 | 4  |
| <b>step 1</b>             |                       | sort 10,11,36 and 15 |    |    |    |   |    |    |    |    |
|                           |                       |                      | 10 | 30 | 11 | 1 | 15 | 31 | 36 | 4  |
| <b>step 2</b>             |                       | sort 30,1,31 and 4   |    |    |    |   |    |    |    |    |
|                           |                       |                      | 10 | 1  | 11 | 4 | 15 | 30 | 36 | 31 |

Figure 73

III Do the following:

$$k = k/2 = 2/2 = 1$$

This is the last pass and will always be an insertion pass.

| <b><math>k = 2</math></b> |                       | List      | 10 | 30 | 11 | 1  | 36 | 31 | 15 | 4 |
|---------------------------|-----------------------|-----------|----|----|----|----|----|----|----|---|
| pass 3                    | <b><math>k</math></b> | $k/2 = 1$ |    |    |    |    |    |    |    |   |
|                           |                       |           | 10 | 30 | 11 | 1  | 36 | 31 | 15 | 4 |
| <b>step 1</b>             |                       | 1         | 4  | 10 | 11 | 15 | 30 | 31 | 36 |   |

\* The last Pass is always an insertion Pass..

Figure 74

**Question: Write code to implement shell sort algorithm.**

Answer The following steps will be involved:

Step1: Define the `shell_sort()` function to sort the list. It will take the list(`input_list`) as input value.

```
def shell_sort(input_list):
```

Step2:

Calculate size,  $n = \text{len}(\text{input\_list})$

Number of steps for the while loop,  $k = n/2$

```
def shell_sort(input_list): n  
= len(input_list)  
k = n//2
```

Step 3:

- While  $k > 0$ :
  - o for  $j$  in range 0, size of input list
    - for  $i$  in range  $(k, n)$ 
      - if the value of element at  $i$  is less than the element located at index  $i-k$ , then swap the two values
    - o set  $k = k//2$

```
while k > 0:  
for j in range(n):  
for i in range(k,n):  
temp = input_list[i]  
if input_list[i] < input_list[i-k]: input_list[i] =  
input_list[i-k] input_list[i-k] = temp  
k = k//2
```

Step 4: Print the value of the sorted list.

Code

```
def shell_sort(input_list):  
n = len(input_list)  
k = n//2  
while k > 0:  
for j in range(n):  
for i in range(k,n):  
temp = input_list[i]  
if input_list[i] < input_list[i-k]: input_list[i] =  
input_list[i-k] input_list[i-k] = temp  
k = k//2  
print(input_list)
```

### Execution

```
shell sort([10,30,11,1,36,31,15,4])
```

### Output

```
[1, 4, 10, 11, 15, 30, 31, 36]
```

### Quick Sort

- Σ In quicksort we make use of a pivot to compare numbers.
- Σ All items smaller than the pivot are moved to its left side and all items larger than the pivot are moved to the right side. This would provide left partition that has all values less than the pivot and right partition having all values greater than the pivot.
- Σ Let's take a list of 9 numbers: [15, 39, 4, 20, 50, 6, 28, 2, 13].
- Σ The last element '13' is taken as the pivot.
- Σ We take the first element '15' as the left mark and the second last element '2' as the right mark.
- Σ If leftmark > pivot and rightmark < pivot then swap left mark and right mark and increment leftmark by 1 and decrement rightmark by 1.
- Σ If leftmark > pivot and rightmark > pivot then only decrement the rightmark.
- Σ Same way if leftmark < pivot and rightmark < pivot then only increment the leftmark.
- Σ If leftmark < pivot and rightmark > pivot, increment leftmark by 1 and decrement right mark by 1.
- Σ When left mark and rightmark meet at one element, swap that element with the pivot.

I

The updated list is now [2, 6, 4, 13, 50, 39, 28, 15, 20]:

- Σ Take the elements to the left of 13, taking 4 as pivot and sort them in the same manner.
- Σ Once the left partition is sorted take the elements on the right and sort them taking 20 as pivot.

## 270 - Python Interview Questions

| quicksort |    |   |    |    |    |    |    |    |            |     |         |  |
|-----------|----|---|----|----|----|----|----|----|------------|-----|---------|--|
| 15        | 39 | 4 | 20 | 50 | 6  | 28 | 2  | 13 | 15>13      | yes | swap    |  |
| 2         | 39 | 4 | 20 | 50 | 6  | 28 | 15 | 13 | 2<13       | yes |         |  |
| 2         | 39 | 4 | 20 | 50 | 6  | 28 | 15 | 13 | 39>13      | yes | no swap |  |
| 2         | 39 | 4 | 20 | 50 | 6  | 28 | 15 | 13 | 28<13      | no  |         |  |
| 2         | 6  | 4 | 20 | 50 | 39 | 28 | 15 | 13 | 39>13      | yes | swap    |  |
| 2         | 6  | 4 | 20 | 50 | 39 | 28 | 15 | 13 | 6<13       | yes |         |  |
| 2         | 6  | 4 | 20 | 50 | 39 | 28 | 15 | 13 | 4>13       | no  | no swap |  |
| 2         | 6  | 4 | 20 | 50 | 39 | 28 | 15 | 13 | 50<13      | no  |         |  |
| 2         | 6  | 4 | 20 | 50 | 39 | 28 | 15 | 13 | index meet |     | swap    |  |
| 2         | 6  | 4 | 13 | 50 | 39 | 28 | 15 | 20 |            |     |         |  |

Figure 75

II

| quicksort |   |   |    |    |    |    |    |    |                                     |     |                                    |  |
|-----------|---|---|----|----|----|----|----|----|-------------------------------------|-----|------------------------------------|--|
| 2         | 6 | 4 | 13 | 50 | 39 | 28 | 15 | 20 | 2>4                                 | yes | No swap                            |  |
| 2         | 4 | 6 | 13 | 50 | 39 | 28 | 15 | 20 | 6<4                                 | no  | pointers meet at 6 swap 6 and 4    |  |
| 2         | 4 | 6 | 13 | 50 | 39 | 28 | 15 | 20 | 50>20                               | yes | swap                               |  |
| 2         | 4 | 6 | 13 | 15 | 39 | 28 | 50 | 20 | 15<20                               | no  |                                    |  |
| 2         | 4 | 6 | 13 | 15 | 20 | 28 | 50 | 39 | 39>20                               | yes | pointers meet at 39 swap 28 and 20 |  |
| 2         | 4 | 6 | 13 | 15 | 20 | 28 | 50 | 39 | 28>39                               | NO  |                                    |  |
| 2         | 4 | 6 | 13 | 15 | 20 | 28 | 39 | 50 | pointers meet at 50 swap with pivot |     |                                    |  |

Figure 76

**Question:** Write the code to implement quick sort algorithm.

**Answer:**

Step: Decide upon the Pivot

- This function takes three parameters, the `list(input_list)`, starting (`fast`) and ending (`last`) index for the list that has to be sorted.
- Take the `input_list.pivot = input_list[last]`. We set the pivot to last value of the list.
- Set the `left_pointer` to first.
- Set `right_pointer` to `last - 1`, because the last element is the pivot.
- Set `pivot_flag` to True.
- While `pivot_flag` is true:
  - If `leftmark > pivot` and `rightmark < pivot` then swap `left mark` and `right mark` and increment `leftmark` by 1 and decrement `rightmark` by 1.

- o If `leftmark > pivot` and `rightmark > pivot` then only decrement the `rightmark`.
- o Same way if `leftmark < pivot` and `rightmark < pivot` then only increment the `leftmark`.
- o If `leftmark < pivot` and `rightmark > pivot`, increment `leftmark` by 1 and decrement `right mark` by 1.
- o When left mark and rightmark meet at one element, swap that element with the pivot.
- o When, `leftmark >= rightmark`, swap the value of the pivot with the element at `left pointer`, set the `pivot_flag` to false.

```
def find_pivot(input_list, first,last):
    pivot = input_list[last]
    print("pivot =", pivot)
    left_pointer = first
    print("left pointer = ", left_pointer, " ,input_list[left_pointer]")
    right_pointer = last-1
    print("right pointer = ", right_pointer, " ,input_list[right_pointer]")
    pivot_flag = True

    while pivot_flag:
        if input_list[left_pointer]>pivot:
            if input_list[right_pointer]<pivot:
                temp      =      input_list[right_pointer]
                input_list[right_pointer]=input_list[left_pointer]
                input_list[left_pointer]= temp
                right_pointer = right_pointer-1
                left_pointer = left_pointer+1

            else:
                right_pointer = right_pointer-1
                left_pointer = left_pointer+1
                right_pointer = right_pointer-1
                if left_pointer >= right_pointer:
                    temp = input_list[last]
```

```
    input_list[last]      =      input_list[left_pointer]
input_list[left_pointer] = temp
pivot_flag = False
print(left_pointer)
return left_pointer
```

Step 2:

Define quicksort(input\_list) function.

- This function will take a list as input.
- Decides the starting point(0) and end point(length\_of\_the\_list-1) for sorting.
- Call the qsHelper() function.

```
def quickSort(input_list):
    first = 0
    last = len(input_list)-1
    qsHelper(input_list,first,last)
```

Step 3:

Define the qsHelper() function

This function checks the first index and last index value, it is a recursive function, it calls the `find_pivot` where leftmark is incremented by 1 and rightmark is decremented by 1. So, as long as the leftmark(which is parameter first in this case) is less than the rightmark(which is parameter last in this case) the while loop will be executed where qsHelper finds new value of pivot, creates ;eft and right partition and calls itslef.

```
def qsHelper(input_list,first,last):
    if first<last:
        partition = find_pivot(input_list,first,last)
        qsHelper(input_list,first,partition-1)
        qsHelper(input_list,partition+1,last)
```

## Code

```

def find_pivot(input_list, first,last):
    pivot = input_list[last]
    left_pointer = first
    right_pointer = last-1
    pivot_flag = True

    while pivot_flag:
        if input_list[left_pointer]>pivot:
            if input_list[right_pointer]<pivot:
                temp = input_list[right_pointer]
                input_list[right_pointer]=input_list[left_pointer]
                input_list[left_pointer]= temp

                right_pointer = right_pointer-1
                left_pointer = left_pointer+1
            else:
                right_pointer = right_pointer-1

        else:
            if input_list[right_pointer]<pivot:
                left_pointer = left_pointer+1
            else:
                left_pointer = left_pointer+1
                right_pointer = right_pointer-1
                if left_pointer >= right_pointer:
                    temp = input_list[last]
                    input_list[last] = input_list[left_pointer]
                    input_list[left_pointer] = temp
                    pivot_flag = False
                    return left_pointer
            def quickSort(input_list):
                first = 0
                last = len(input_list)-1
                qsHelper(input_list,first,last)

            def qsHelper(input_list,first,last):
                if first<last:
                    partition      =      find_pivot(input_list,first,last)
                    qsHelper(input_list,first,partition-1)
                    qsHelper(input_list,partition+1,last)

```

Execution

```
input_list=[15,39,4,20,50,6,28,2,13]
quickSort(input_list)
print(input_list)
```

Output

```
[2, 4, 6, 13, 15, 20, 28, 39, 50]
```



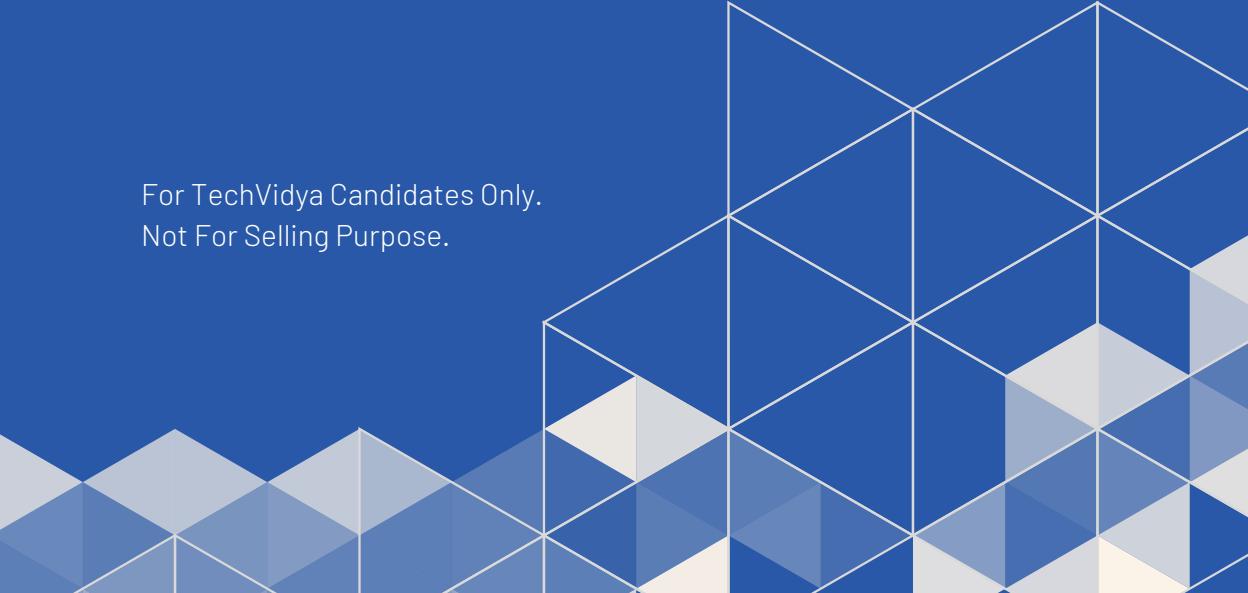


# TECHVIDYA

ISO 9001:2015 Accredited Company

*"Stay Updated, Stay Ahead"*

For TechVidya Candidates Only.  
Not For Selling Purpose.



# PYTHON INTERVIEW QUESTIONS - 2

## "STAY UPDATED, STAY AHEAD"

TechVidya is ISO Certified 9001:2015 accredited EdTech Company registered with ROC under the companies act 1956, offers self-paced, online and offline programs. TechVidya pedagogy is rooted in the principle that every young mind should be equipped with exceptional knowledge and skills that benefit them in real life.

**YEAR  
2023**

---

Company Info

**TechVidya**  
EdTech Company

Company Website

**techvidya.education**  
Official Website

Company Contact

**+91 83759 66700**  
Official Number

# Python Interview Questions

---

*Brush up for Your Next Python Interview  
with 240+ Solutions on Most Common  
Challenging Interview Questions*

---

# Python Solved Questionnaire

## Learning objective:

This book will help you to learn:

- The core concept of Python
- 

The OOPs concept

- Modules in Python
- Python GUI (Tkinter)
- File Handling
- Python database
- NumPy, Pandas
- Django, Flask

Let us begin!

### 1. What is the history behind Python?

Python was released in 1991 by Guido van Rossum. The history behind the name is, in the 1970s, there was a popular BBC comedy TV show called Monty Python's Fly Circus

*Van Rossum*

happened to be the big fan of that show. So, when Python was developed, Rossum named the project *Python*.

## 2. What is Python?

**Ans.** Python is a high-level, interpreted, interactive, and object-oriented scripting language. It is designed to be highly readable:

- Points to know about Python -It supports functional and structured programming methods as well as OOP.
- It supports automatic garbage collection.
- It can be used as a scripting language and can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- i. • It can be easily integrated with
  - ii. C
  - iii. , C++,
  - iv. COM,
  - v. ActiveX,
  - vi. CORBA,
  - Java.

## 3. In which application area Python can be used?

**Ans.** Python can be built in the following areas:

- GUI-based desktop applications
- Web applications
- Games
- Scientific and computational applications
- Language development
- Enterprise and business applications development
- Operating systems

## 4. What are the benefits of Python?

**Ans.** Here are the benefits of Python:

**Open source**: Python language is developed under an OSI-approved open source license, which makes

it free to use and distribute, including for commercial purposes.

- **Easy learning:** Python offers excellent readability and uncluttered simple-to-learn syntax which helps beginners to utilize this programming language.
- **Extensive support library:** Python provides a large standard library that includes areas such as Internet protocols, string operations, web services tools, and operating system interfaces. It reduces the length of code to be written significantly.
- **User-friendly data structure:** Python has built-in list and dictionary data structures that can be used to construct fast runtime data structures.
- **Productivity and speed:** Python has clean object-oriented design, provides enhanced process control capabilities and possesses strong integration and text processing capabilities and its unit testing framework, all of which contribute to the increase in its speed and productivity.

## 5. Explain memory management in Python?

**Memory** management is the process of efficiently allocating, de-allocating, and coordinating memory so that all the different processes run smoothly and can optimally access different system resources. Memory management also involves cleaning the memory of objects that are no longer being accessed.

In Python, the memory manager is responsible for these kinds of tasks by periodically running to clean up, allocate, and manage the memory. Unlike C, Java, and other programming languages, Python manages objects by using reference counting. This means that the memory manager keeps track of the number of references to each object in the program.

Lets understand memory management by following points:

- Python memory is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have access to this private heap and the interpreter takes care of this Python private heap.

- The allocation of Python heap space for Python objects is done by the Python memory manager. The core API gives access to some tools for the programmer to code.
- The Python memory manager manages chunks of memory called “*Blocks*”. A collection of blocks of the same size makes up the “*Pool*”. Pools are created on Arenas, chunks of 256kB memory allocated on `heap=64` pools. If the objects get destroyed, the memory manager fills this space with a new object of the same size.
- Python also has an inbuilt garbage collector, which recycles all the unused memory and frees the memory and makes it available to the heap space.

## **6. What are the different environment variables in Python? And what's the use of these variables?**

**Ans.** Here are the different environment variables and its uses:

- PYTHONPATH: It is the same as the PATH variable. The Python interpreter uses it to search the module files.
- PYTHONSTARTUP: It stores the path of an initialization script containing the Python code. It gets to run every time the Python interpreter starts.
- PYTHONCASEOK: In Windows, it makes the Python find the first case-insensitive match in an import statement. You need to set it for activation.

## **7. Is Python Scripting language?**

**Ans.** Python is a scripting language. It is also an interpreted and high-level programming language for the purpose of general programming requirements. It was designed and developed by the Software Developer named Guido van Rossum.

It was first released in the year 1991. It is a dynamic type of discipline and has strong typing too. Filename extensions for python scripting language are of different types such as `.py`, `.pyc`, `.pyd`, `.pyo`, `.pyw`, `.pyz`.

## **8. List the datatypes supported by Python?**

**Ans.** Here are the datatypes:

- **Text type:** str
- **Numeric types:** int, float, complex
- **Sequence types:** list, tuple, range
- **Mapping type:** dict
- **Set types:** set, frozenset
- **Boolean type:** bool
- **Binary types:** bytes, bytearray, memoryview

### 9. What is the output of the following?

```
str="swati
computers"?      print
```

**Ans.** (str) It would print the complete string “swati computers”

### 10. What is the output of the following

```
str="swati
computers"?      print
```

**Ans.** It would print “s”

### 11. What is the output of the following

```
str= "swati computers!" ?
print str* 2
```

**Ans.** It will print the string two times. Swati computers!
Swati computers!

### 12. What is the output of the following?

```
list  = [109,  'swati']?
print list * 2
```

**Ans.** It will print the list two times. Output would be
['swati', 109, 'swati']

### 13. How to check keywords in Python?

**Ans.** Type the following code:

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

**14. What is pep 8?**

**Ans.** PEP stands for Python Enhancement Proposal. It is a set of rules that specify how to format Python code for maximum readability.

**15. What is the output of following?**

```
str="{s}{c}{j}".  
format(j='Jaipur',s='Swati',c='Computers')  
print(str)
```

Ans. SwatiComputersJaipur

**16. What will be the output of the following?**

```
str="apple#banana#kiwi#oran  
ge" print(str.split("#",2))
```

Ans. ['apple', 'banana', 'kiwi#orange']

**17. What are python modules? Name some commonly used built-in modules in Python?**

**Ans.** Python modules are files containing Python code. This code can either be function classes or variables. A Python module is a .py file containing executable code. Some of the commonly used built-in modules are:

- Os
- sys
- math
- random
- data time
- json

**18. What are the Local and Global variables in Python?**

**Ans.** Global variables are defined outside of any module and function. These variables can be accessed by any function in the program. Local variables are defined inside the function where it is used.

## 19. What is the meaning of type conversion in Python?

**Ans.** Type conversion refers to the conversion of one data type into another:

- int(): Converts any data type into the integer type.
- float(): Converts any data type into float type.
- ord(): Converts characters into integer.
- hex(): Converts integers to hexadecimal.
- oct(): Converts an integer to the octal.
- tuple(): This function is used to convert to a tuple.
  - set(): This function returns the type after converting to set.
  - list(): This function is used to convert any data type to a list type.
  - dict(): This function is used to convert a tuple of order (key, value) into a dictionary.
- str(): Used to convert an integer into a string.
  - complex(real,imag): This function converts real numbers to complex(real,imag) number.

### Example: 1

```
num_int = 100
```

```
num_flo = 100.23
```

```
num_new = num_int + num_flo
```

```
print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))
```

```
print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

### Solution:

```
datatype of num_int: <class 'int'>
```

```
datatype of num_flo: <class 'float'>
```

```
Value of num_new: 200.23000000000002
```

```
datatype of num_new: <class 'float'>
```

**Example: 2**

```
print('ASCII value of "S" is: ' + str(ord('S')))  
print('Hexadecimal value of 253 is: ' +  
str(hex(253)))  
print('Octal value of 57 is: ' + str(oct(57)))  
print('Binary value of 54 is: ' + str(bin(54)))
```

**Solution:**

ASCII value of "S" is: 83  
Hexadecimal value of 253 is: 0xfd  
Octal value of 57 is: 0o71  
Binary value of 54 is: 0b110110

**Note: You can try on Jupyter notebook for every datatype given.**

**20. Explain List in Python?**

**Ans.** Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of the different data types. The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 at the beginning of the list and working their way to end -1.

**Note: Please check Q. 22 for example**

**21. What do the signs + and \* mean in List in Python?**

**Ans.** The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator.

**22. Give an example of Python List?**

**Ans.**

```
list = [ 'swati', 109 ,]  
  
print (list ) # Prints complete list  
print (list[0])# Prints first element of the list  
print (list[1:3])# Prints elements starting from 2nd
```

```
till 3rd
print (list[2:])# Prints elements starting from 3rd
element
print (list* 2)# Prints list two times
print (list + list)# Prints concatenated lists
```

### 23. What are the tuples?

**Ans.** A tuple is a datatype similar to the Python list. It consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses () .

**Note:** Please check Q.26 for example

### 24. Difference between List and tuple?

**Ans.** The following table shows the required difference:

| List                                | Tuple                      |
|-------------------------------------|----------------------------|
| It is mutable in nature             | It is immutable            |
| Values in lists are enclosed in [ ] | Values are enclosed in ( ) |

*Table 1*

**Note: Everything in Python is Object.** Mutable objects can be changed after it is created while the immutable object cannot change. Objects of built-in types like int, float, bool, str, tuple, unicode) are immutable. Objects of built-in types like (list, set, dict) are mutable. Custom classes are generally mutable.

### 25. What is the use of zip () in python ?

**Ans. zip()** The returns an iterator and takes iterable as argument. These iterables can be list, tuple, dictionary etc. It maps similar index of every iterable to make a single entity.

**Example 1:**

```
name=["swati","shweta"]
age=[10,20]
new_entity=zip(name,age)
new_entity=set(new_entity)
print(new_entity)
```

The output is {('shweta', 20), ('swati', 10)}

### **Example 2 :**

```
a=["1","2","3"]
b=["a","b","c"]
c=[x+y for x, y in zip(a,b)]
print(c)
```

The output is: ['1a', '2b', '3c']

If the passed iterators have different number of argument then the iterator with the least items decides the length of the new iterator.

### **Example 3:**

```
name=["swati","shweta","Sweety"]
age=[10,20]
new_entity=zip(name,age)
new_entity=set(new_entity)
print(new_entity)
```

The output is: {('shweta', 20), ('swati', 10)}

## **26. Example of the tuple**

**? Ans.**

```
tuple = ( 'swati', 109 , 1 )
```

```
print (tuple )# Prints complete list
print (tuple[0])# Prints first element of the list
print (tuple[1:3])# Prints elements starting from
2nd till 3rd
print (tuple[2:]) # Prints elements starting from
3rd element
print (tuple * 2 )# Prints list two times
print (tuple + tuple)# Prints concatenated lists
```

The output is as follows:

```
In [5]: tuple = ( 'swati', 109 , 1 )

print( tuple      )
print (tuple[0]  )
print (tuple[1:3])
print (tuple[2:]  )
print( tuple * 2 )
print (tuple + tuple)

('swati', 109, 1)
swati
(109, 1)
(1,)
('swati', 109, 1, 'swati', 109, 1)
('swati', 109, 1, 'swati', 109, 1)
```

**Figure 1**

## 27. What is the dictionary in Python?

**Ans.** Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. Dictionaries are enclosed by curly braces ({} ) and values can be assigned and accessed using square braces ([]).

**Note: Please check example 29**

## 28. Create a two (multi) dimensional list?

**Ans.**

```
two_dim_list = [[0] * 3]*3
```

```
print(two_dim_list)
```

```
two_dim_list[0][0] = 1
```

```
two_dim_list[0][1] = 2
```

```
two_dim_list[0][2] = 3
```

```
two_dim_list[1][0] = 4
```

```
two_dim_list[1][1] = 5
```

```
two_dim_list[1][2] = 6
```

```
two_dim_list[2][0] = 7  
two_dim_list[2][1] = 8  
two_dim_list[2][2] = 9  
print(two_dim_list)
```

Take a look at the following output:

```
In [14]: two_dim_list = [[0] * 3]*3  
  
print(two_dim_list)  
  
two_dim_list[0][0] = 1  
two_dim_list[0][1] = 2  
two_dim_list[0][2] = 3  
  
two_dim_list[1][0] = 4  
two_dim_list[1][1] = 5  
two_dim_list[1][2] = 6  
  
two_dim_list[2][0] = 7  
two_dim_list[2][1] = 8  
two_dim_list[2][2] = 9  
  
print(two_dim_list)
```

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]  
[[7, 8, 9], [7, 8, 9], [7, 8, 9]]
```

*Figure 2*

## 29. Example of the dictionary

**? Ans.**

```
dict = {}
```

```
dict['one'] = "This is one"
```

```
dict[2] = "This is two"
```

```
keyVal = {'name': 'swati', 'age': 10, 'dept':  
'software'}
```

```
print(dict['one']) # Prints value for 'one' key  
print(dict[2]) # Prints value for 2 key  
print(keyVal) # Prints complete dictionary
```

```
print(keyVal.keys()) # Prints all the keys
print(keyVal.values()) # Prints all the values
```

The output is as follows:

```
In [6]: dict = {}
dict['one'] = "This is one"
dict[2]      = "This is two"

keyVal = {'name': 'swati', 'age':10, 'dept': 'software'}


print(dict['one'])           # Prints value for 'one' key
print(dict[2])              # Prints value for 2 key
print(keyVal)               # Prints complete dictionary
print(keyVal.keys())         # Prints all the keys
print(keyVal.values())       # Prints all the values


This is one
This is two
{'name': 'swati', 'age': 10, 'dept': 'software'}
dict_keys(['name', 'age', 'dept'])
dict_values(['swati', 10, 'software'])
```

*Figure 3*

### 30. Write a program to convert a string into int?

**Ans.**

```
age = "18"

print(age)

# Converting string to integer

int_age = int(age)

print(int_age)
```

### 31. What is frozenset() in Python?

**Ans.** A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. The `frozenset()` is an inbuilt function in Python that takes an iterable object as input and makes them immutable. In Python, `frozenset` is the same as `set` except its elements are immutable.

```
# tuple of numbers
num = (1, 2, 3, 4, 5)

# converting tuple to frozenset
fnum = frozenset(num)
# printing details
print("frozenset Object is : ", fnum)
```

**Answer:** frozenset Object is :frozenset({1, 2, 3, 4, 5})

### 32. Example to convert int to string?

**Ans.**

```
a=10
# printing integer converting to string
c = str(a)
print ("After converting integer to string :
",end="")
print (c)
```

**Answer:** After converting integer to string : 10

### 33. What is \_\_\_\_\_ name \_\_\_\_\_ in Python?

**Ans.** Python has no inbuilt main function to start the execution of the program. Python has special variable `__name__`; it provides the functionality of the main function. Here is the output:

```
def func1():
    print ("The value of __name__ is " + __name__)
if __name__ == '__main__':
    func1()
```

```
The value of __name__ is __main__
```

*Figure 4*

### 34. Explain different operators in Python

**? Ans**The Python language supports the following types of operators:

- **Arithmetic operators:** + , - , \* , / , \*\* (Exponent) , // floor division , % (Modulus)

**For example:**

- $9//2 = 4$  and  $9.0//2.0 = 4.0$ ,  $-11//3 = -4$ ,
- $-11.0//3 = -4.0$
- **Relational operators :**< , > , <= , >= , != , <> , ==
- **Assignment operators:** = , += , -= , \*= , /= , // = , %=
- **Logical operators:** and, or, not

**Bitwise operators:** &, | , ~ , << , >> , ^

**Membership operators:** Python's membership operators test for membership in a sequence, such as

| Operator | Description                                                                                                                                           | Example                                                                                       |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| in       | Evaluates to true if it finds x in y, herein results in a 1 if a variable in the specified x is a member of sequence y. sequence and false otherwise. | x in y, here results in a 1 if y. sequence and false otherwise.                               |
| not in   | Evaluates to true if it does not find a variable in the specified sequence and member of sequence y. false otherwise.                                 | x not in y, here “not in” results in a 1 if x is not a member of sequence y. false otherwise. |

**Table 2**

```

a = 10
b = 20
list = [1, 2, 3, 4, 5];
if ( a in list ):
    print( "Line 1 - a is available in the given list")
else:
    print ("Line 1 - a is not available in the given
list")
if ( b not in list ):
    print ("Line 2 - b is not available in the given
list")
else:
    print ("Line 2 - b is available in the given list")

```

- **Identity operators:** Identity operators compare the memory locations of two objects.

| Operator | Description                                                                                                     | Example                                                            |
|----------|-----------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| is       | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here are results in 1 if id(x) equals id(y).               |
| is not   | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here are not results 1 if id(x) is not equal to id(y). |

**Table 3**

```

a = 20
b = 20

if ( a is b ):
    print ("Line 1 - a and b have same identity")
else:
    print( "Line 1 - a and b do not have same identity")

if ( id(a) == id(b) ):
    print ("Line 2 - a and b have same identity")
else:
    print ("Line 2 - a and b do not have same identity")

b = 30

if ( a is b ):
    print ("Line 3 - a and b have same identity")
else:
    print( "Line 3 - a and b do not have same identity")

if ( a is not b ):
    print ("Line 4 - a and b do not have same identity")
else:
    print ("Line 4 - a and b have same identity")

```

When you execute the preceding program, it produces the following result:

- Line 1 - a and b have same identity
- Line 2 - a and b have same identity
- Line 3 - a and b do not have same identity
- Line 4 - a and b do not have same identity

### **35. Write a program to convert all string in the list to an integer?**

**Ans.**

```
test_list = ['10', '14', '1', '6', '7']

# Printing original list
print ("Original list is : " + str(test_list)) # conversion revise, here you need not to
#convert

# perform conversion
For i in range(0, len(test_list)):
    test_list[i] = int(test_list[i])

# Printing modified list
print ("Modified list is : " + str(test_list))
```

The output is as follows:

Original list is: ['10', '14', '1', '6', '7'] Modified list  
is: [10, 14, 1, 6, 7]

### **36. Remove empty strings from list of strings?**

**Ans.**

```
# initializing list
test_list = [ "", "swatiComputers", "", "is", "best",
            ""]

# Printing original list
print ("Original list is : " + str(test_list))

# using remove() to
```

```
# perform removal  
while("") in test_list) :  
    test_list.remove("")  
  
# Printing modified list  
    print ("Modified list is : " + str(test_list))
```

The output is as follows:

Original list is: [", 'swatiComputers', ", 'is', 'best', "]  
Modified list is: ['swatiComputers', 'is', 'best']

### **37. How is Python an interpreted language?**

**Ans.** An interpreter is a kind of program that executes other programs. When you write Python programs, it converts source code written by the developer into the intermediate language which is again translated into the native language/machine language that is executed. The Python code you write is compiled into Python bytecode, which creates a file with extension .pyc. The compilation is simply a translation step, and byte code is a lower-level, and platform-independent, representation of your source code.

The .pyc file, created in the compilation step, is then executed by appropriate virtual machines. The Virtual Machine is the runtime engine of Python and it is always present as part of the Python system and is the component that truly runs the Python scripts. Technically, it's just the last step of what is called the Python interpreter.

### **38. What are the .pyc files?**

**Ans.** The .py files contain the source code of a program, whereas the.pyc file contains the bytecode of your program. We get bytecode after compilation of the .py file (source code). The .pyc files are not created for all the files that you run. It is only created for the files that you import.

### **39. Write a program to ask two numbers and print their addition?**

**Ans.**

```
num1 = int(input("Enter first number))
```

```
num2 = int(input("Enter second number"))

# printing the sum in integer
print(num1 + num2)
```

**40. Write a program to input N numbers in a list?****Ans.**

```
nums= []

# number of elements as input
n = int(input("Enter number of elements : "))

# iterating till the range
for i in range(0, n):

    ele = int(input("Enter number"))
    nums.append(ele) # adding the element

print(nums)
```

Here is the screen with output:

```
In [9]: nums= []

# number of elements as input
n = int(input("Enter number of elements : "))

# iterating till the range
for i in range(0, n):
    ele = int(input())
    nums.append(ele) # adding the element

print(nums)

Enter number of elements : 3
1
2
3
[1, 2, 3]
```

*Figure 5***41. Write a program to find out greater out of three?****Ans.**

```
#input values or assign
```

```
a=2  
n=3  
c=4  
  
#code for check greater number  
  
if (a>b)and (a>c) :  
    print "A is greater"  
elif (b>a) and (b>c) :  
    print "B is greater"  
else:  
    print "C is greater"
```

Or

```
Take input from user  
a=int(input("enter num"))  
b=int(input("enter num"))  
c=int(input("enter num"))
```

#### **42. Write a program to find out prime number between 1 to 100?**

**Ans.**

```
for num in range(1,100):  
    for i in range(2,num):  
        if num%i == 0:  
            j=num/i  
            print ('not prime')  
                break #to move to the next number, the  
#first FOR  
    else:  
        print( num, 'is a prime number')
```

#### **43. Write a program to print values of tuple?**

**Ans.**

```
name = ['swati' , 'sweety' , 'shweta']  
  
for id in range(len(name)):
```

```
print ('Hello :',name[id])
```

The output is as follows:

Hello : swati

Hello : sweety

Hello : shweta

#### 44. Write a program to print letters of the name?

**Ans.**

```
name="swati"
```

```
for i in name :
```

```
    print (i)
```

```
    print ('\n')
```

#### 45. Write the code to display the following pattern:

\*

\*\*

\*\*\*

\*\*\*\*

**Ans.\***

```
for i in range (1:5):
```

```
    for j in range(5,i,-1):
```

```
        print(end=" ")
```

```
    for k in range(1:i):
```

```
        print("* ",end=" ")
```

```
    print("\r")
```

#### 46. Write a code to display the following character pattern:

A

B B

C CC

D DDD

E EEEE

**Ans.**

```

num = 65

for i in range(0, 5):

    for j in range(0, i+1):
        ch = chr(num)

        # printing char value
        print(ch, end=" ")
        num = num + 1

    # ending line after each row
    print("\r")

```

Following image shows the code and output:

```

In [13]: num = 65

for i in range(0, 5):

    for j in range(0, i+1):
        ch = chr(num)

        # printing char value
        print(ch, end=" ")

        num = num + 1

    # ending line after each row
    print("\r")

```

A  
B C  
D E F  
G H I J  
K L M N O

*Figure 6*

#### 47. Write a program to print 1 to 10 using a while loop?

**Ans.**

```

count = 1

while (count < 11):
    print('num:', count)
    count = count + 1

```

#### 48. Explain and Print a random number?

**Ans.** Python number method and range() returns a randomly selected element from range (start, stop, step):

- start: Start point of the range. This would be included in the range.
- stop: Stop point of the range. This would be excluded from the range.
- step: Steps to be added in a number to decide a random number.

```
import random
```

```
# Select an even number in 100 <= number < 1000
```

```
print ("randrange(100, 1000, 2) : ", random.  
randrange(100, 1000, 2))
```

```
# Select another number in 100 <= number < 1000
```

```
print ("randrange(100, 1000, 3) : ", random.  
randrange(100, 1000, 3))
```

The output for the code will be as follows:

```
randrange(100, 1000, 2) : 976
```

```
randrange(100, 1000, 3) : 520
```

#### 49. How to call the base class inherited method in the child class?

**Ans. Using super():** Refer to the following screenshot:

```
In [7]: class Computer():
    def __init__(self,ram,storage):
        print("RAM=",ram)
        print("Storage=",storage)
class Laptop(Computer):
    def __init__(self,ram,storage,model):
        print("Model is :",model)
        super().__init__(ram,storage)

hp=Laptop('8GB','1TB','hp PAV X360')
```

```
Model is : hp PAV X360
RAM= 8GB
Storage= 1TB
```

**Figure 7**

- 50. As in Java “instanceof” is used to check whether a class is subclass of given class or not, which function or keyword is used in Python to check subclass?**

**Ans. Syntax:**  
issubclass(subClassName,superClassName)

**Example:**

```
issubclass(equation,Addition) # it will return true  
issubclass(subtraction,equation) # it will return  
false
```

- 51. Is there any destructor in**

**Python?** Ans. Destructors are called when an object gets destroyed. Python has a garbage collector that handles memory management automatically. The `__del__()` method is known as a destructor method in Python. It is called when all references to the object have been deleted.

**Example:**

```
class Student:
```

```
    def __init__(self):
```

```
        print('welcome student...')
```

```
    def __del__(self):
```

```
        print('Destructor called, student deleted')
```

```
obj = Student()
```

```
#del obj
```

- 52. How does Python Garbage collector works?**

**Ans.** Python deletes unwanted objects (built-in types or class instances) automatically to free the memory space. Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that points changes. An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with `del`, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches

zero, Python collects it automatically. You can also del call () to delete the unused object.

**Example:**

Point is a class, and pt is an object pt  
= Point()  
del pt

**53. Write a code to create a for loop that iterates through a tuple of class objects. Then call the methods without being concerned about which class type each object is?**

**Ans.**

```
class shape:  
    def __init__(self):  
        print("I am a shape")  
  
    class rect(shape):  
        def myself(self):  
            print("Myself rectangle")  
  
    class sqr(shape):  
        def myself(self):  
            print("Myself square")  
  
    r=rect()  
    s=sqr()  
    for sh in (r,s):  
        sh.myself()
```

**54. Give an example of overriding.**

**Ans.**

```
class shape:  
    def __init__(self):  
        print("I am a shape")  
    def area(self):  
        print("Every shape has area")
```

```

class rect(shape):
def myself(self):
print("Myself rectangle")
def area(self):
print("Area of rectangle")

class sqr(shape):
def myself(self):
print("Myself square")
def area(self):
print("Area of square")

r=rect()
s=sqr()

for sh in (r,s):
sh.myself()
sh.area()

```

In [7]:

```

class shape:
    def __init__(self):
        print("I am a shape")
    def area(self):
        print("Every shape has area")

class rect(shape):
    def myself(self):
        print("Myself rectangle")
    def area(self):
        print("Area of rectangle")

class sqr(shape):
    def myself(self):
        print("Myself square")
    def area(self):
        print("Area of square")

r=rect()
s=sqr()

for sh in (r,s):
    sh.myself()
    sh.area()

```

```

I am a shape
I am a shape
Myself rectangle
Area of rectangle
Myself square
Area of square

```

**Note: Overriding is the concept of OOPs(Object-oriented programming).**

**Pillars of OOPs:**

- **Abstraction:** To hide complexity and show which is necessary to a current problem scenario.
- **Encapsulation:** Combine data of the similar type of objects into a single unit.
- **Polymorphism:** Many forms of a single object.

**Polymorphism is of two types:**

- **Early binding:** Example - Overloading
- **Late Binding:** Example - Overriding
- **Inheritance:** Use for code reusability, where child (sub) class can use properties of the parent (base) class.

To know more about OOPs, refer “*Java -A complete practical solution*” by BPB Publications.

**55. Create a class Robot and set the robot name by the setter method, constructor?**

**Ans.**

```
class Robot:

    def __init__(self, name=None):
        self.name = name

    def say_hi(self):
        if self.name:
            print("Hello, I am " + self.name)
        else:
            print("Hello, I am a robot without a name")

    def set_name(self, name):
        self.name = name

    def get_name(self):
        return self.name
```

```
x = Robot("Swati")
y = Robot()
z = Robot()

y.set_name("Swati")

x.say_hi()
y.say_hi()
z.say_hi()
```

```
.2]: class Robot:

    def __init__(self, name=None):
        self.name = name

    def say_hi(self):
        if self.name:
            print("Hello, I am " + self.name)
        else:
            print("Hello, I am a robot without a name")

    def set_name(self, name):
        self.name = name

    def get_name(self):
        return self.name

x = Robot("Swati")
y = Robot()
z = Robot()

y.set_name("Swati")

x.say_hi()
y.say_hi()
z.say_hi()
```

```
Hello, I am Swati
Hello, I am Swati
Hello, I am a robot without a name
```

---

*Figure 9*

**56. Create a queue in Python and print all its elements. Ans.**

```
import queue

# Queue is created as an object 'obj'
obj = queue.Queue(maxsize=10)
```

```
# Data is inserted in 'obj' at the end using put()
obj.put(19)
obj.put(64)
obj.put(73)
obj.put(46)

#while not obj.empty:
#print(obj.get())

print(obj.get())
print(obj.get())
print(obj.get())
print(obj.get())
```

The output is as follows:

19  
64  
73  
46

## 57. What is GUI programming? How to create GUI applications in Python?

**Ans.** Modern applications are not restricted to console input/output, they have a user-friendly graphical user interface. These applications can receive inputs through mouse clicks and can enable the user to choose from alternatives with the help of radio buttons, dropdown lists, and other GUI elements (or widgets). Such applications are developed using one of the various graphics libraries available.

A graphics library is a software toolkit having a collection of classes that define the functionality of various GUI elements. These graphics libraries are generally written in C/C++.

Python provides various options for developing GUI, one of them is Tkinter. Python provides the standard library Tkinter for creating the graphical user interface for desktop-based applications.

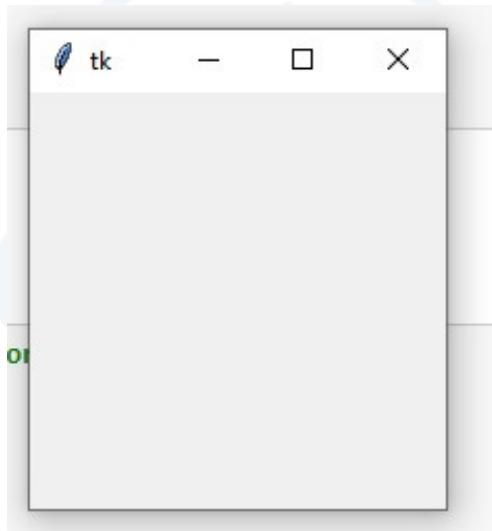
Here are the steps to create a Tkinter window:

- i. Import the Tkinter module.
- ii. Create the main application window.
- iii. Add widgets such as labels, buttons, and frames to the window.
- iv. Call the main event loop so that the actions can take place on the user's computer screen.

**Example:**

```
from tkinter import *
top=Tk()
top.mainloop()
```

Here's the screenshot:



*Figure 10*

**58. Explain Tkinter widgets.**

| SN | Widget | Description                                                                |
|----|--------|----------------------------------------------------------------------------|
| 1  | Button | Click Button. The button is used to add buttons to the Python application. |
| 2  | Canvas | The Canvas widget is used to draw the canvas on the window.                |

|    |             |                                                                                                                                                      |
|----|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3  | Checkbutton | The Checkbutton is used to display the CheckBox on the window.                                                                                       |
| 4  | Entry       | The entry widget is used to display the single-line text field to the user. It is commonly used to accept user values.                               |
| 5  | Frame       | It can be defined as a container to which, another widget can be added and organized.                                                                |
| 6  | Label       | A label is a text used to display some message or information about the other widgets.                                                               |
| 7  | ListBox     | The ListBox widget is used to display a list of options to the user.                                                                                 |
| 8  | Menubutton  | The Menubutton is used to display the menu items to the user.                                                                                        |
| 9  | Menu        | It is used to add menu items to the user.                                                                                                            |
| 10 | Message     | The Message widget is used to display the message-box to the user.                                                                                   |
| 11 | Radiobutton | The Radiobutton is different from a checkbutton. Here, the user is provided with various options and the user can select only one option among them. |
| 12 | Scale       | It is used to provide the slider to the user.                                                                                                        |
| 13 | Scrollbar   | It provides the scrollbar to the user so that the user can scroll the window up and down.                                                            |
| 14 | Text        | It is different from entry because it provides a multi-line text field to the user so that the user can write the text and edit the text inside it.  |
| 14 | Toplevel    | It is used to create a separate window container.                                                                                                    |
| 15 | Spinbox     | It is an entry widget used to select from options of values.                                                                                         |
| 16 | PanedWindow | It is like a container widget that contains horizontal or vertical panes.                                                                            |
| 17 | LabelFrame  | LabelFrame is a container widget that acts as the container.                                                                                         |
| 18 | MessageBox  | This module is used to display the message-box in the desktop-based applications.                                                                    |

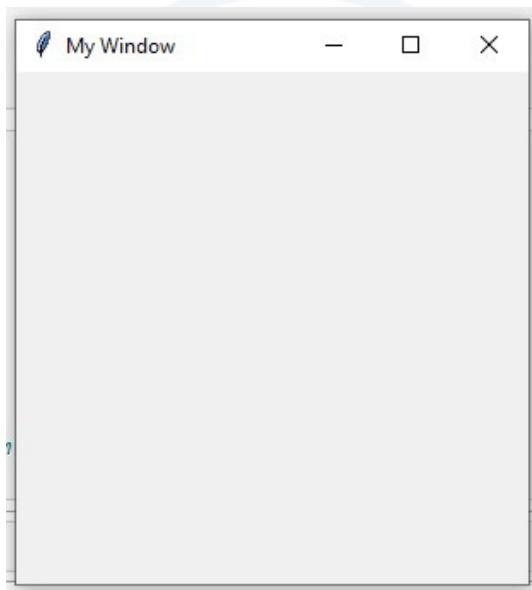
**Table 4**

- 59. Create a Tkinter window and change its title, height, width , position from the top and left.**

**Ans.**

```
from tkinter import *
top = Tk()
top.title("My Window")
top.geometry("300x300+200+200")
top.mainloop()
#top.geometry(width x height + from left + from top)
```

The following is the output:



**Figure 11**

- 60. How to restrict the window resize at runtime?**

**Ans.**

**Function:**

```
Resizable (true/false, true / false)
from tkinter import *
top = Tk()
```

```
top.title("My Window")
top.geometry("300x300+200+200")
top.resizable(0,0)
top.mainloop()
```

**61. Change the background color of the Tkinter window.**

**Ans.**

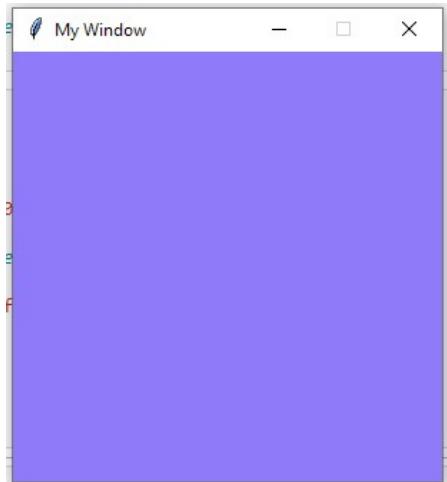
```
from tkinter import *
top = Tk()
top.title("My Window")
top.geometry("300x300+200+200")

# top.configure(bg='blue')

top['background']='#856ff8'

top.resizable(0,0)
top.mainloop()
```

Here is the screenshot:

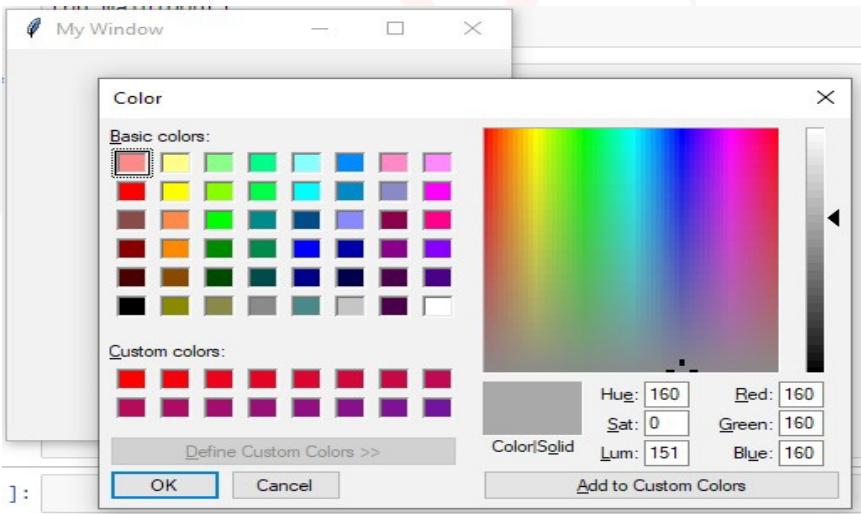


**Figure 12**  
TechVidya Career Private Limited  
"Stay Updated, Stay Ahead"

**62. Open the color picker dialog.****Ans.**

```
from tkinter import *
from tkinter.colorchooser import *
top = Tk()
top.title("My Window")
top.geometry("300x300+200+200")
color=askcolor()
print(color[0])
#top['background']="#856ff8"
top.resizable(0,0)
top.mainloop()
```

Take a look at the following image:



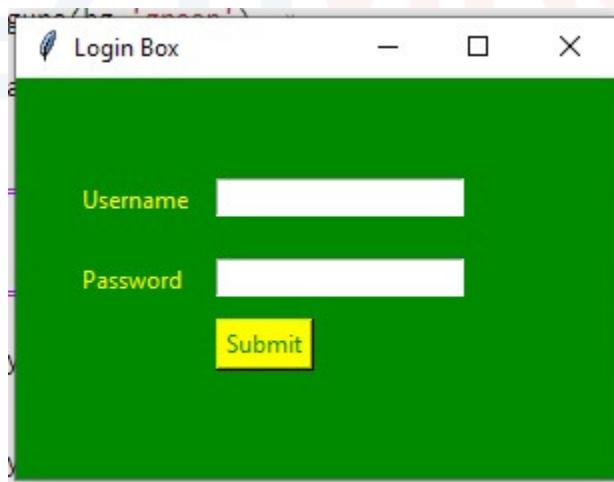
**Figure 13**

**63. Create a login box in Tkinter Python.****Ans.**

```
from tkinter import *
from tkinter.colorchooser import *
```

```
top = Tk()  
  
top.title("Login Box")  
top.geometry("300x200+200+200")  
top.configure(bg='green')  
  
uname = Label(top, text =  
"Username",fg='yellow',bg='green').place(x = 30,y =  
50)  
  
#creating label  
password = Label(top, text =  
"Password",bg='green',fg='yellow').place(x = 30, y =  
90)  
  
submitbtn = Button(top, text = "Submit",bg= "yellow",  
fg= "green").place(x = 100, y = 120)  
e1 = Entry(top,width = 20).place(x = 100, y = 50)  
  
e2 = Entry(top, width = 20).place(x = 100, y = 90)  
top.mainloop()
```

Here is the output screenshot:



**Figure 14**

**64. Create a frame to add and multiply two given numbers.****Ans.**

```
import tkinter as tk
from functools import partial

def addi(answer, n1, n2):
    num1 = (n1.get())
    num2 = (n2.get())
    result = int(num1)+int(num2)
    answer.config(text="Result = %d" % result)
    return

def multi(answer, n1, n2):
    num1 = (n1.get())
    num2 = (n2.get())
    result = int(num1)*int(num2)
    answer.config(text="Result = %d" % result)
    return

root = tk.Tk()
root.geometry('400x200+100+200')
root.title('Calculator')

number1 = tk.StringVar()
number2 = tk.StringVar()

labelNum1 = tk.Label(root, text="A").grid(row=1,
column=0)
labelNum2 = tk.Label(root, text="B").grid(row=2,
column=0)
answer = tk.Label(root)
answer.grid(row=7, column=2)
entryNum1 = tk.Entry(root, textvariable=number1).
grid(row=1, column=2)
entryNum2 = tk.Entry(root, textvariable=number2).
grid(row=2, column=2)
addi= partial(addi, answer, number1, number2)
```

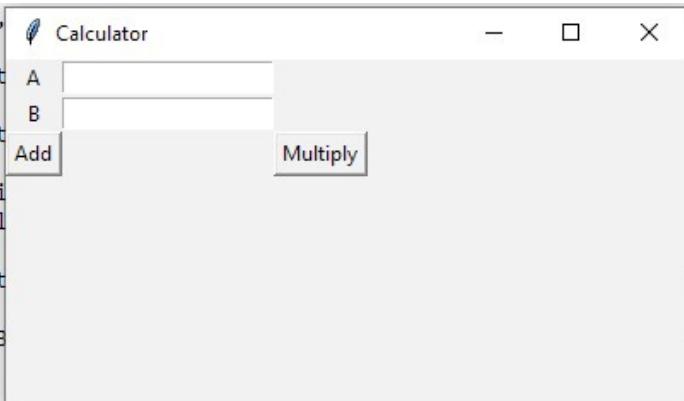
```

multi= partial(multi, answer, number1, number2)

buttonAdd = tk.Button(root, text="Add",
command=addi).grid(row=3, column=0)
buttonMulti = tk.Button(root, text="Multiply",
command=multi).grid(row=3, column=10)
root.mainloop()

```

Here is the screenshot:



**Figure 15**

## 65. What is the use of 'partial' in the preceding example?

**Ans.**

**As discussed in above program:**

```

addi= partial(addi, answer, number1, number2)
multi= partial(multi, answer, number1, number2)

```

```

buttonAdd = tk.Button(root, text="Add",
command=addi).grid(row=3, column=0)
buttonMulti = tk.Button(root, text="Multiply",
command=multi).grid(row=3, column=10)

```

Partial is being used as one strategy for passing the parameters into the function you'd like the button to do (by not having parameters at all, and building a new function that just has the parameters baked into it).

If you do not need to pass an argument, avoid using partial.

**Example:**

```
def disp():
    answer.config(text="Hello user ")
    return

buttonMulti = tk.Button(root, text="Msg",
    command=disp).grid(row=3, column=7)
```

**66. Write a program to get selected fruit from the listbox in Python?****Ans.**

```
from tkinter import *
from functools import partial

top = Tk()

def display(answer,mylist):
    a=mylist.get(ACTIVE)
    answer.config(text=a)
    return

sb = Scrollbar(top)
sb.pack(side = RIGHT, fill = Y)
Lb=Label(top,text="select Favourite Fruit")
mylist = Listbox(top, yscrollcommand = sb.set )
answer=Label(top)

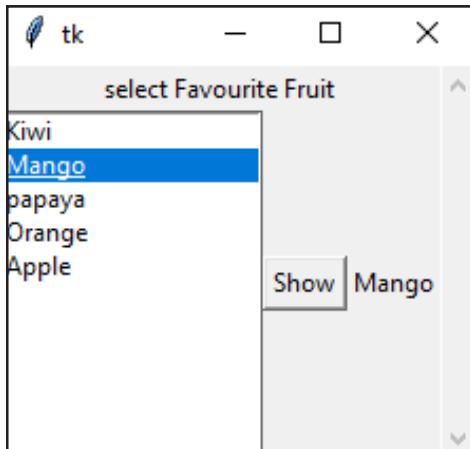
display=partial(display,answer,mylist)
b=Button(top,text="Show",command=display)

mylist.insert(1,"Kiwi")
mylist.insert(2,"Mango")
mylist.insert(3,"papaya")
mylist.insert(4,"Orange")
mylist.insert(5,"Apple")

Lb.pack()
mylist.pack( side = LEFT )
sb.config( command = mylist.yview )
```

```
b.pack(side=LEFT)
answer.pack(side=RIGHT)
mainloop()
```

Here is the screenshot:



*Figure 16*

**67. Write a program to create three checkboxes for languages such as English, Hindi, French, and two radio buttons for gender selection, and on clicking on the thanks button, the window should close.**

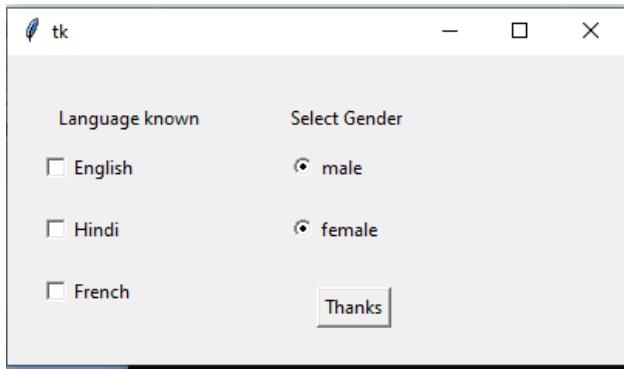
**Ans.**

```
from tkinter import *
top = Tk()
top.geometry("400x200")
L=Label(top,text="Language known").place(x=30,y=30)
c1=Checkbutton(top,text="English").place(x=20,y=60)
c2=Checkbutton(top,text="Hindi").place(x=20,y=100)
c3=Checkbutton(top,text="French").place(x=20,y=140)
L=Label(top,text="Select Gender").place(x=180,y=30)
r1=Radiobutton(top,text="Male").place(x=180,y=60)
```

```
r2=Radiobutton(top,text="female").place(x=180,y=100)

b=Button(top,text="Thanks",command=top.destroy).
place(x=200,y=150)
top.mainloop()
```

Here is the output:



*Figure 17*

### 68. Write a program to select radio buttons from different groups.

**Ans.**

```
from tkinter import *

top = Tk()
top.geometry("400x200")

var=StringVar()
var1=StringVar()

L=Label(top,text="Qualification").place(x=30,y=30)

r1 = Radiobutton(top, text='Graduate', variable=var,
value='A').place(x=30,y=50)
r2 = Radiobutton(top, text='Under Graduate',
variable=var, value='B').place(x=30,y=80)
r3 = Radiobutton(top, text='Post Graduate',
variable=var, value='C').place(x=30,y=110)
L=Label(top,text=" Gender").place(x=180,y=30)
```

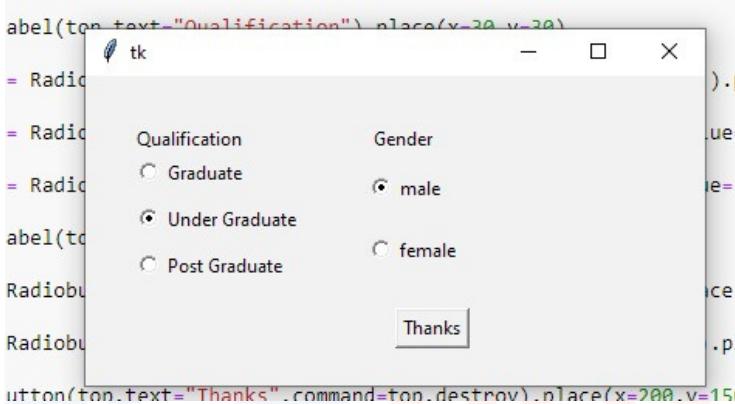
```
r1=Radiobutton(top,text="male",variable=var1,value='male').place(x=180,y=60)

r2=Radiobutton(top,text="female",variable=var1,value='female').place(x=180,y=100)

b=Button(top,text="Thanks",command=top.destroy).place(x=200,y=150)

top.mainloop()
```

The following image is the output:



**Figure 18**

## 69. Create a scale to select the value from 10 to 100

**00. Ans.**

```
from tkinter import *

def sel():
    selection = "Value = " + str(var.get())
    label.config(text = selection)

root = Tk()
root.geometry("300x300")
var = DoubleVar()

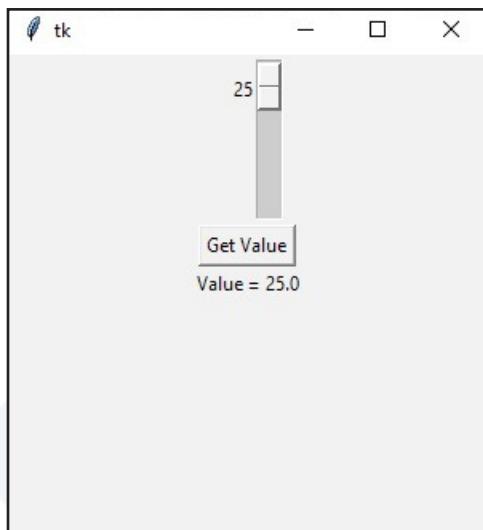
scale = Scale( root, variable = var,from_=10,to=1000
)
```

```
scale.pack(anchor=CENTER)

button = Button(root, text="Get Value", command=sel)
button.pack(anchor=CENTER)

label = Label(root)
label.pack()
root.mainloop()
```

The following is the output screenshot:



*Figure 19*

**70. Write a program to change the width of the label by changing the scale value?**

**Ans.**

```
from tkinter import *

def sel():
    selection = "Value = " + str(var.get())
    w= int(var.get())
    label.config(width=w)

    root = Tk()
    root.geometry("300x300")
```

```
var = DoubleVar()  
  
scale = Scale( root, variable = var,from_=10,to=1000  
)  
scale.pack(anchor=CENTER)  
  
button = Button(root, text="Change Width",  
command=sel)  
button.pack(anchor=CENTER)  
  
label = Label(root,bg="RED")  
label.pack()  
  
root.mainloop()
```

The output screenshot is as follows:

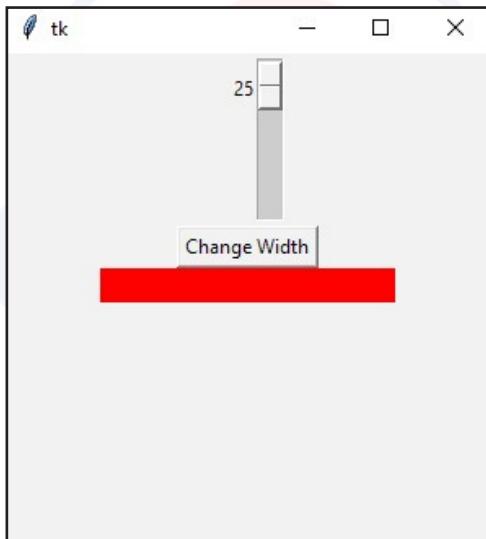


Figure 20

**71. Write a program to change label text on the select radio button?**

**Ans.**

```
import tkinter as tk  
from functools import partial  
  
top=tk.Tk()
```

```
top.geometry("300x300")

r11=tk.StringVar()

def gen():
# print(r11.get())
a=r11.get()
label.config(text=a)

#gen=partial(gen,L)

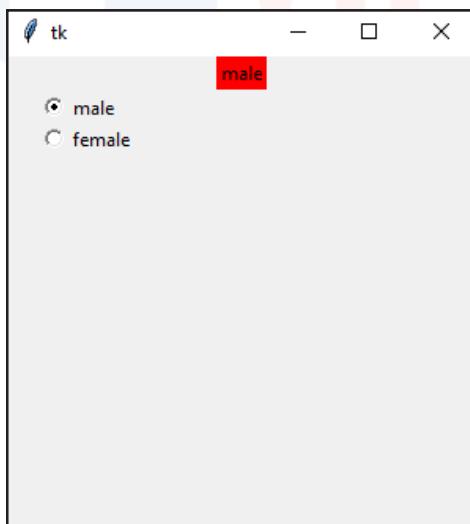
r1=tk.Radiobutton(top,text="male",value="male",
command=gen,variable=r11).place(x=20,y=20)

r2=tk.Radiobutton(top,text="female",value="female",
variable=r11,command=gen).place(x=20,y=40)

label = tk.Label(top,bg="RED")
label.pack()

top.mainloop()
```

You will get the following screen:



**Figure 21**

## 72. List options of Tkinter Entry.

**Ans.** The following table lists the options:

| SNO | Option               | Description                                                                                                                                                                                                                |
|-----|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | Bg                   | The background color of the widget.                                                                                                                                                                                        |
| 2   | Bd                   | The border width of the widget in pixels.                                                                                                                                                                                  |
| 3   | Cursor               | The mouse pointer will be changed to the cursor type set to the arrow, dot, etc.                                                                                                                                           |
| 4   | Export selection     | The text written inside the entry box will be automatically copied to the clipboard by default. We can set exportselection to 0 to not copy this.                                                                          |
| 5   | Fg                   | It represents the color of the text.                                                                                                                                                                                       |
| 6   | Font                 | It represents the font type of the text.                                                                                                                                                                                   |
| 7   | Highlight background | It represents the color to display in the traversal highlight region when the widget does not have the input focus.                                                                                                        |
| 8   | Highlight color      | It represents the color to use for the traversal highlight rectangle that is drawn around the widget when it has the input focus.                                                                                          |
| 9   | Highlight thickness  | It represents a non-negative value indicating the width of the highlight rectangle to draw around the outside of the widget when it has the input focus.                                                                   |
| 10  | Insert background    | It represents the color to use as a background in the area covered by the insertion cursor. This color will normally override either the normal background for the widget.                                                 |
| 11  | Insert borderwidth   | It represents a non-negative value indicating the width of the 3-D border to draw around the insertion cursor. The value may have any of the forms acceptable to Tk_GetPixels.                                             |
| 12  | Insert offtime       | It represents a non-negative integer value indicating the number of milliseconds the insertion cursor should remain off in each blink cycle. If this option is zero, then the cursor doesn't blink: it is on all the time. |
| 13  | Insert ontime        | Specifies a non-negative integer value indicating the number of milliseconds the insertion cursor should remain on in each blink cycle.                                                                                    |

|    |                   |                                                                                                                                             |
|----|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 14 | Insertwidth       | It represents the value indicating the total width of the insertion cursor. The value may have any of the forms acceptable to Tk_GetPixels. |
| 15 | Justify           | It specifies how the text is organized if the text contains multiple lines.                                                                 |
| 16 | Relief            | It specifies the type of the border. Its default value is FLAT.                                                                             |
| 17 | Select background | The background color of the selected text.                                                                                                  |
| 18 | Selectborderwidth | The width of the border to display around the selected task.                                                                                |
| 19 | Select foreground | The font color of the selected task.                                                                                                        |
| 20 | Show              | It is used to show the entry text or some other type instead of the string. For example, the password is typed using stars (*).             |
| 21 | Textvariable      | It is set to the instance of the StringVar to retrieve the text from the entry.                                                             |
| 22 | Width             | The width of the displayed text or image.                                                                                                   |
| 23 | Xscroll command   | The entry widget can be linked to the horizontal scrollbar if we want the user to enter more than the actual width of the widget.           |

Table 5

**73. Create a Notepad window in Tkinter.****Ans.**

```
from tkinter import Button, Tk, Menu ,Label ,Text

top = Tk()
def command():
    master2 = Tk()
    master2.geometry("300x300+0+2")
    master2.resizable(0,0)
    master2.title("New Window")
    #label = Label(master2, text="This is the new
    #window")
    #label.pack()
    text=Text(top).place(x=0,y=0)
```

```
master2.mainloop()
menubar = Menu(top)
file = Menu(menubar, tearoff=0)
file.add_command(label="New", command=command)
file.add_command(label="Open")
file.add_command(label="Save")
file.add_command(label="Save as...")
file.add_separator()
file.add_command(label="Close")

file.add_command(label="Exit", command=top.quit)

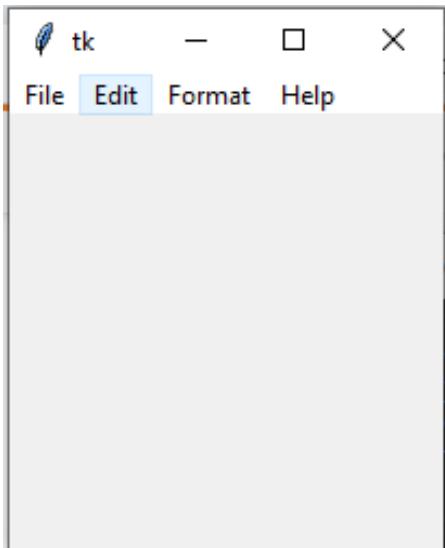
menubar.add_cascade(label="File", menu=file)
edit = Menu(menubar, tearoff=0)
edit.add_command(label="Undo")

edit.add_command(label="Cut")
edit.add_command(label="Copy")
edit.add_command(label="Paste")
edit.add_separator()
edit.add_command(label="Delete")
edit.add_command(label="Select All")
menubar.add_cascade(label="Edit", menu=edit)

format=Menu(menubar)
format.add_command(label="font")
format.add_command(label="wordWrap")
menubar.add_cascade(label="Format",menu=format)
help = Menu(menubar, tearoff=0)
help.add_command(label="About Notepad")
menubar.add_cascade(label="Help", menu=help)

top.config(menu=menubar)
top.mainloop()
```

The following is the output:



*Figure 22*

#### **74. Open Gmail on a button click.**

**Ans.**

```
from tkinter import Button, Tk, Menu ,Label ,Text
```

```
top = Tk()
```

```
def mail():
```

```
    import webbrowser
```

```
    webbrowser.open('https://gmail.com',new=1)
```

```
    menubar = Menu(top)
```

```
    file = Menu(menubar, tearoff=0)
```

```
    file.add_command(label="Open Gmail",command=mail)
```

```
    menubar.add_cascade(label="Gmail", menu=file)
```

```
    top.config(menu=menubar)
```

```
    top.mainloop()
```

The following screen will be visible:

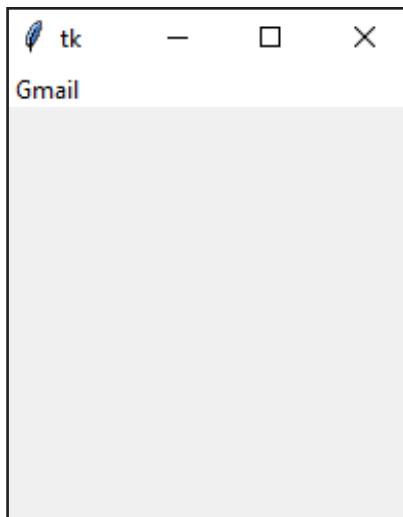


Figure 23

**75. Write a program to calculate age, create a spin box or entry box to select the day, month and year, then subtract the year from the current year.**

**Ans.**

```
from tkinter import *
from datetime import date
top = Tk()

def age():
    a=IntVar()
    b=IntVar()
    a=int(spin2.get())
    b=int(today.year)
    c=b-a
    print(c)
    msg=Label(top,text="Age : "+str(c))
    msg.pack()
    top.geometry("400x240")
```

```
today = date.today()

label=Label(top,text="select DOB")
label1=Label(top,text="select Day")
spin = Spinbox(top, from_= 1, to = 31)

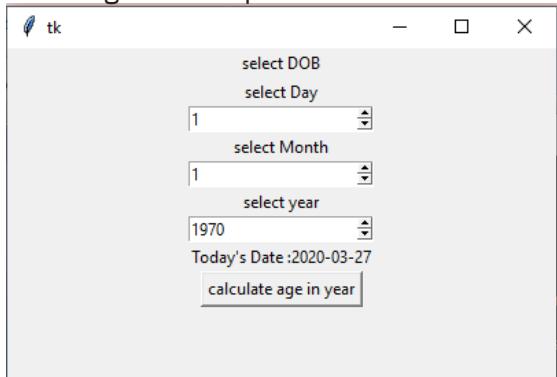
label2=Label(top,text="select Month")
spin1 = Spinbox(top, from_= 1, to = 12)
label3=Label(top,text="select year")
spin2 = Spinbox(top, from_= 1970, to = 2020)
label.pack()
label1.pack()
spin.pack()
label2.pack()
spin1.pack()
label3.pack()
spin2.pack()

label4=Label(top,text="Today's Date :" +str(today))
label4.pack()

b=Button(top,text="calculate age in
year",command=age)
b.pack()

top.mainloop()
```

The following is the output screenshot:



**Figure 24**

**76. Write a program to open different color windows on the click of the button?**

**Ans.**

```
from tkinter import *
class A:
    def CB(self):
        B()
    def CC(self):
        C()
    def CD(self):
        D()
    def __init__(self):
        top = Tk()
        top.title("My Window")
        top.geometry("300x300+200+200")
        # top.configure(bg='blue')
        top['background']='#856fff8'
        bt=Button(top,text="Green",command=self.CB)
        bt.pack()
        bt1=Button(top,text="Blue",command=self.CC)
        bt1.pack()
        bt2=Button(top,text="Red",command=self.CD)
        bt2.pack()
        top.resizable(0,0)
        top.mainloop()
class B(object):
    def __init__(self):
        top = Tk()
        top.title("My Window")
        top.geometry("300x300+200+200")
        # top.configure(bg='red')
        top['background']='#678423'
        top.resizable(0,0)
```

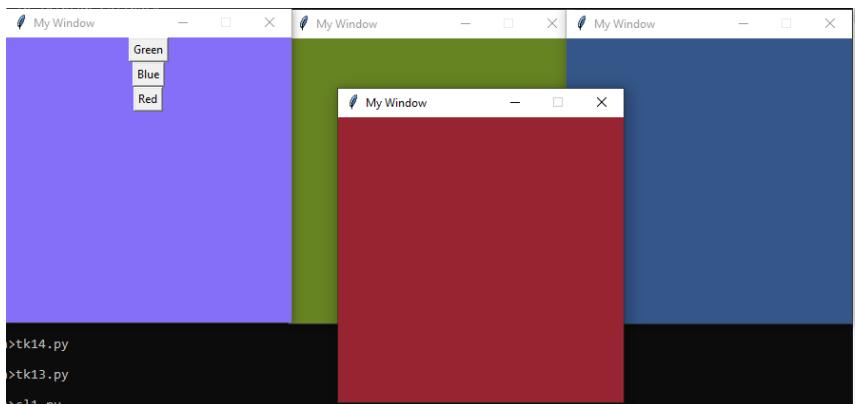
```
top.mainloop()

class C(object):
    def __init__(self):
        top = Tk()
        top.title("My Window")
        top.geometry("300x300+200+200")
        # top.configure(bg='red')
        top['background']='#345689'
        top.resizable(0,0)
        top.mainloop()

class D(object):
    def __init__(self):
        top = Tk()
        top.title("My Window")
        top.geometry("300x300+200+200")
        # top.configure(bg='red')
        top['background']='#982432'
        top.resizable(0,0)
        top.mainloop()

def main():
    A()
    main()
```

The following is the output screen:



**Figure 25**  
TechVidya Career Private Limited  
"Stay Updated, Stay Ahead"

**77. Write a program to select an item from the combo box and add it to the listbox. You can remove items from the list box.**

**Ans.**

```
from tkinter import *
from tkinter.ttk import Combobox
window=Tk()

data=("Apple", "Kiwi", "Papaya",
"Grapes","Banana","Orange")

def add():
    a=StringVar()
    a=cb.get()
    lb.insert(END,a)

def remove():
    a=lb.curselection()
    lb.delete(a[0])

var = StringVar()

lbl=Label(window,text="Select favourite fruit").
place(x=150,y=20)
bt=Button(window,text=">>",command=add).
place(x=200,y=70)
bt=Button(window,text="<<",command=remove).
place(x=200,y=110)
lb=Listbox(window, height=5, selectmode='multiple')

lb.place(x=250, y=50)

cb=Combobox(window,
values=data) cb.place(x=30, y=50)

window.title('Hello Python')
window.geometry("400x300+10+10")
window.mainloop()
```

The following is the output:

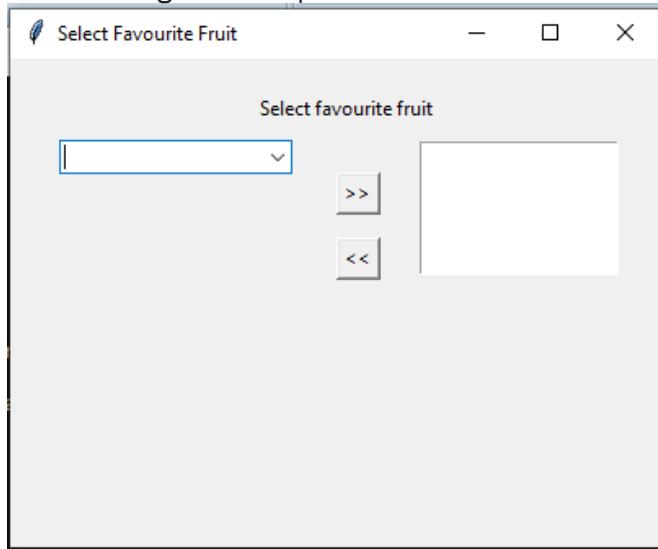


Figure 26

### 78. Is indentation required in Python?

**Ans.** Indentation is necessary for Python. It specifies a block of code. All code within loops, classes, functions, etc. is specified within an indented block. It is usually done using four space characters. If your code is not indented necessarily,

### 79. How is Multithreading achieved in Python?

**Ans.**

- Python has a multi-threading package but if you want to multi-thread to speed up your code, then it's usually not a good idea to use it.
- Python has a construct called the **Global Interpreter Lock (GIL)**. The GIL makes sure that only one of your 'threads' can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread.
- This happens very quickly so to the human eye, it may seem like your threads are executing in parallel, but they are just taking turns using the same CPU core.

- All this GIL passing adds overhead to execution. This means that if you want to make your code run faster, then using the threading package often isn't a good idea.

**80. Write some disadvantages of multi-threading? (Multi-threading is a concept and same in all language)****Ans.**

- On a single processor system, multithreading won't hit the speed of computation. The performance may downgrade due to the overhead of managing threads.
- Synchronization is needed to prevent mutual exclusion while accessing shared resources. It directly leads to more memory and CPU utilization.
- Multithreading increases the complexity of the program, thus also making it difficult to debug.
- It raises the possibility of potential deadlocks.
- It may cause starvation when a thread doesn't get regular access to shared resources. The application would then fail to resume its work.

**81. Write a program to create a popup menu?****Ans.**

```
from tkinter import *  
  
top = Tk()  
  
w = Label(top, text="Right-click to display menu",  
width=40, height=20)  
w.pack()  
  
# create a menu  
popup = Menu(top, tearoff=0)  
popup.add_command(label="Cut")  
popup.add_command(label="Copy")  
popup.add_command(label="Paste")  
popup.add_separator()  
popup.add_command(label="Select All")
```

```

popup.add_command(label="Delete")

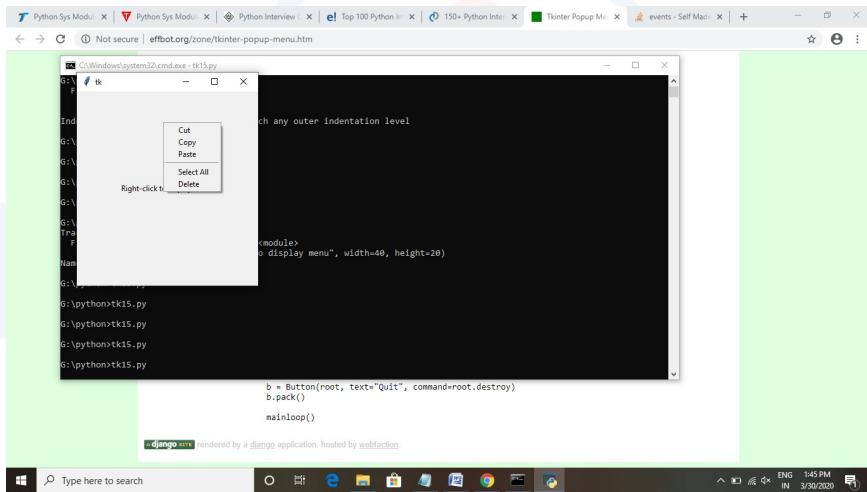
def do_popup(event):
    # display the popup menu
    try:
        popup.tk_popup(event.x_root, event.y_root,
                      0)
    finally:
        popup.grab_release()

w.bind("<Button 3>", do_popup)

mainloop()

```

Take a look at the following output screen:



**Figure 27**

## 82. What is the Dogpile effect?

**Ans.** The Dogpile effect is referred to as the event when the cache expires, and websites are hit by the multiple requests made by the client at the same time. This effect can be prevented by using a semaphore lock. In this system when value expires, the first process acquires the lock and starts generating new value.

### 83. What is a Frame in Python GUI?

**Ans.** A Frame in Python can be related as a storage holder for other Graphical User Interface or GUI elements such as Label, Text Entry, Text Box, Check Button, RadioButton, etc.

The following is the output screen:

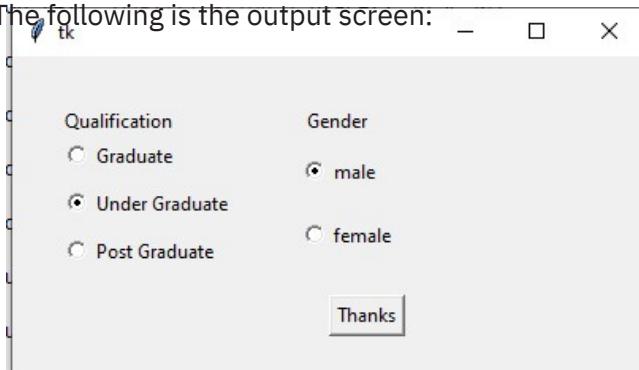


Figure 28

### 84. What is the difference between the input() method and the raw\_input() method?

**Ans.** The raw\_input() method returns string values whereas the input() method returns integer values. The input() method was used in Python 2.x versions whereas Python 3.x and later versions use the raw\_input() method. However, the input() method has been replaced by the raw\_input() method in Python 3.x.

### 85. What is the difference between a Lambda and Def?

**Ans.** A Def is a function that can contain multiple expressions whereas a Lambda can contain only one single expression.

A Def method can contain return statements whereas a Lambda cannot contain return statements. A Lambda can be used inside lists and dictionaries.

#### Example of Lambda:

```
x = lambda a: a + 10
print(x(5))
```

#### Example of def:

```
def addi(answer, n1, n2):
```

```
num1 = (n1.get())
num2 = (n2.get())
result = int(num1)+int(num2)
answer.config(text="Result = %d" % result)
return
```

## 86. How do you define the dimensions of a window in a Python Graphics Program?

**Ans.** It can be defined using the geometry() method. It takes in two parameters: width and height respectively.

Example: geometry("width \* height")

## 87. What is a DocString in Python and what is it used for?

**Ans.** A DocString represents Document String that is used to Document Python Modules, Classes, and Methods.

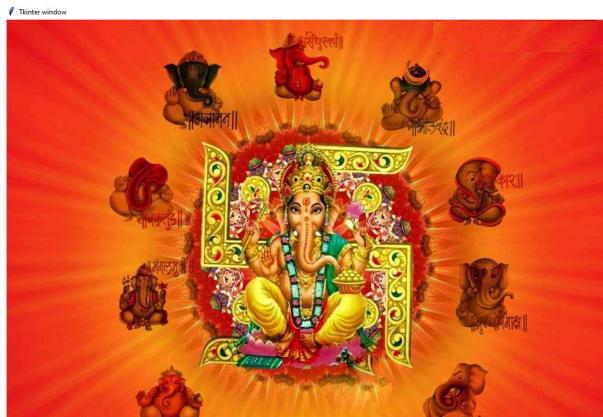
## 88. How to add an image in Tkinter?

**Ans.**

```
from tkinter import *
from PIL import Image, ImageTk
class Window(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.pack(fill=BOTH, expand=1)
        load = Image.open("1.jpg")
        render = ImageTk.PhotoImage(load)
        img = Label(self, image=render)
        img.image = render
        img.place(x=0, y=0)
    root = Tk()
app = Window(root)
```

```
root.wm_title("Tkinter window")
root.geometry("400x420")
root.mainloop()
```

The following is the output:



*Figure 29*

**89. List any six Python Library.**

**Ans.** Here are six Python Library:

- Pandas
- Numpy
- Keras
- Eli5
- SciPy
- Theano

**90. If someone wants to halt the program flow and display text, how to add a delay in execution?**

**Ans.** The Sleep function plays a very important role in a situation where we want to halt the program flow and let other executions happen. It basically adds a delay to the execution and it will only pause the current thread and not the whole program:

```
import time
sleep_time = 1
print('Hello')
```

```
time.sleep(sleep_time)
print("Students!")
time.sleep(sleep_time)
print("Hope you all are doing well!")
```

## 91. Explain threads in Python.

Threads are a single execution part of a program. Several threads can run concurrently, they share the same memory space and are lightweight processes and, they do not require much memory space. It can temporarily be put on hold (also known as **sleeping**) while other threads are running (called **yielding**).

The methods provided by the Thread class are as follows:

- **run()**: The run() method is the entry point for a thread.
- **start()**: The start() method starts a thread by calling the run method.
- **join([time])**: The join() waits for threads to terminate.
- **isAlive()**: The isAlive() method checks whether a thread is still executing.
- **getName()**: The getName() method returns the name of a thread.
- **setName()**: The setName() method sets the name of a thread.

### Example:

```
import threading
import time

exitFlag = 0

class myThread (threading.Thread):

    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
```

```
def run(self):
    print ("Starting " + self.name)
    print_time(self.name, 5, self.counter)
    print ("Exiting " + self.name)

def print_time(threadName, counter, delay):
    while counter:
        if exitFlag:
            threadName.exit()
        time.sleep(delay)
        print ("%s: %s" % (threadName, time.ctime(time.time())))
        counter -= 1

# Create new threads
thread1 = myThread(1, "First", 1)
thread2 = myThread(2, "Second", 2)

# Start new Threads
thread1.start()
thread2.start()
```

## 92. Write a program to check whether a phone number is valid or not using regX? It should contain only ten digits.

**Ans.**

```
import re

phone = "1234567890"

if re.search('\d{10}', phone):
    print ("Phone Num : ", phone)
else:
    print("invalid")
```

Take a look at the following output:

```
import re

phone = "1234567890"
#phone = "123567890"

if re.search('\d{10}', phone):
    print ("Phone Num : ", phone)
else:
    print("invalid")
```

Phone Num : 1234567890

*Figure 30*

**93. Validate e-mail using regular expression in Python.** Ans.

```
import re

email = "swaticomputers01@gmail.com"

if re.search('^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$', email):
    print("Correct")
else:
    print("invalid")
```

The output will be:

Correct

**94. What is Jython?**

**Ans.** Jython is a Java implementation of Python. Jython is freely available for both commercial and non-commercial use and is distributed with source code under the PSF License v2.

**95. Why Jython?**

**Ans.**

- **Rapid application development:** Python programs are typically 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.

- **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.

Some of the projects where Jython is embedded are TigerJython, IBM Websphere, Apache PIG, etc. To embed Python code into Java, you need to import required libraries.

Download <https://www.jython.org/download> and work on it.

## 96. What is wxPython?

**Ans.** wxPython is a cross-platform GUI toolkit for the Python programming language. It allows Python programmers to create programs with a robust, highly functional graphical user interface, simply, and easily. wxPython is an Open source and a cross-platform toolkit. This means that the same program will run on multiple platforms without modification. Currently, supported platforms are Microsoft Windows, Mac OS X and macOS, and Linux or other Unix-like systems with GTK2 or GTK3 libraries.

## 97. What is PyCharm and wxPython?

**Ans.** Pycharm is an editor and integrated development environment by Jetbrains.com. Wxpython is a GUI library used for making GUI apps written in Python/C++.

## 98. List some GUI frameworks for Python?

**Ans.**

|                |                                                                                                                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Tkinter</b> | Tkinter is commonly bundled with Python, using Tk and is Python's standard GUI framework. It is popular for its simplicity and graphical user interface. It is open source and available under the Python License. |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                 |                                                                                                                                                                                                                                                                             |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>wxPython</b> | WxPython is an open-source wrapper for cross-platform GUI library WxWidgets (earlier known as WxWindows) and implemented as a Python extension module. With WxPython, you as a developer can create native applications for Windows, Mac OS, and Unix.                      |
| <b>PyGUI</b>    | PyGUI is a graphical application cross-platform framework for Unix, Macintosh, and Windows. It is simple and lightweight.                                                                                                                                                   |
| <b>PySide</b>   | PySide is a free and cross-platform GUI toolkit. PySide currently supports Linux/X11, Mac OS X, Maemo, and Windows, and support for Android is in the plans for the near future. PySide provides tools to work with multimedia, XML documents, network, databases, and GUI. |
| <b>PyQt</b>     | PyQt is a cross-platform Python GUI Framework implementing the Qt library for the Qt (owned by Nokia) application development framework. Currently, PyQt is available for Unix/Linux, Windows, Mac OS X, and Sharp Zaurus.                                                  |
| <b>Kivy</b>     | Kivy is an OpenGL ES 2 accelerated framework for the creation of new user interfaces. It supports multiple platforms namely Windows, MacOSX, Linux, Android-iOS, and Raspberry Pi. It is an open source.                                                                    |

**Table 6**

Qt initiated and sponsored by Nokia, Qt is a UI framework and a cross-platform application.

## 99. What is the need for File Handling?

**Ans.** Sometimes, data may be large enough and we need to store for future reference. Then we use the local file system to store data so that it can be accessed in the future. Just like in C and other programming languages you can create, read, or write files.

### Syntax:

```
file_object = open(file_name [, access_mode][,
buffering])
```

**100. Explain the File access mode. Ans.**

| SN | Access mode | Description                                                                                                                                                                                 |
|----|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | R           | It opens the file to read-only. The file pointer exists at the beginning.                                                                                                                   |
| 2  | Rb          | It opens the file to read-only in the binary format.                                                                                                                                        |
| 3  | r+          | It opens the file to read and write both.                                                                                                                                                   |
| 4  | rb<br>+     | It opens the file to read and write both in the binary format.                                                                                                                              |
| 5  | W           | It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name.                                                     |
| 6  | W           | It opens the file to write only in the binary format.                                                                                                                                       |
| 7  | b<br>w<br>+ | It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. |
| 8  | wb+         | It opens the file to write and read both in the binary format.                                                                                                                              |
| 9  | A           | It opens the file in the append mode.                                                                                                                                                       |
| 10 | A<br>b      | It opens the file in the append mode in the binary format.                                                                                                                                  |
| 11 | a+          | It opens a file to append and read both.                                                                                                                                                    |
| 12 | ab+         | It opens a file to append and read both in the binary format.                                                                                                                               |

**Table 7**

**101. Write a program to read a line from an existing file?**

**Ans.**

```
# Open a file
fo = open("tk21.py", "r")

str = fo.read(10);
print ("Read String is : ", str)

fo.close()
```

Take a look at the following screenshot:

```
G:\python\pycharm\venv\Scripts\python.exe G:/python/pycharm/ww.py
Read String is :  import re

Process finished with exit code 0
```

**Figure 31**

## 102. Write a program to read the complete file?

**Ans.**

```
# Open a file
fo = open("tk21.py", "r")
str = fo.read(10)
while (str):
    print (str)
    str = fo.read(10)

fo.close()
```

The output is as follows:

```
G:\python\pycharm\venv\Scripts\python.exe G:/python/pycharm/ww.py
import re

phone = "
swaticompu
ters01@gma
il.com"

#
Remove an
ything oth
er than di
gits
if re
.search('^
```

```
\w+([\.-]?
\w+)*@\w
+( [\.-]?
\w+) *
(\.\w{2,3
})+$', pho
ne):
p
rint("Phon
e Num :",
phone)
el
se:
pr
int("inval
id")
```

Process finished with exit code 0

**103. Write a program to read five names from the console and write it into the file?**

**Ans.**

```
# Open a file
fo = open("name.txt", "w")

for i in range (1,6):

    print ("Enter name")
    name=input()
    fo.write( "Name\t"+name+"\n")

# Close open file
file fo.close()
```

The output is as follows:

Name Swati

Name

Sweety

Name Shweta

Name Nidhi

Name Aavya

**104. Which Python module is used to create a new folder, removing, and renaming the file?**

**Ans.** osOS    The Python module is used. The module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality.

```
import os  
print(os.name)
```

nt

*Figure 32*

**105. Write the output of an existing file into another file without using read/write operations of file?**

**Ans.** tk21.py to capture writing the output of txt . The Python code is in tk24.py:

Tk21.py

import re

email = "swaticomputers01@gmail.com"

```
if re.search('^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$', email):  
    print("Email : ", email)
```

else:

print("invalid")

tk24.py

import subprocess

with open("output.txt", "wb") as f:

```
subprocess.check_call(["python", "tk21.py"],  
stdout=f)
```

### Note: Difference between “open” and “with open” ?

The open method syntax:

```
file_handle=open("name","mode")
```

with open syntax:

with open ("name of file" , "mode" ) as keyword :

.....

.....

Open methods take two arguments and you have to explicitly close the file after read/write operation, if you forget to close the garbage collector will close, but we are not sure about this. While with open will close, the file itself after the with block close:

**output.txt**

Email: swaticomputers01@gmail.com

- 106. Create a GUI form (Tkinter), enter user name and password, and save them into a file.**

**Ans.**

```
from tkinter import *
from tkinter.colorchooser import *
from functools import partial

top = Tk()

def xx(a, b):
    u = a.get()
    p = b.get()
    f = open("users.txt", "a")
    f.write("Uname: " + u + "\tPwd: " + p + "\n")
    f.close()
    a.set("")
    b.set("")

top.title("Login Box")
top.geometry("300x200+200+200")
```

```
top.configure(bg='blue')

uname = Label(top, text="Username", fg='white',
bg='blue').place(x=30, y=50)
a = StringVar()
b = StringVar()
password = Label(top, text="Password", fg='white',
bg='blue').place(x=30, y=90)
e1 = Entry(top, width=20, textvariable=a).
place(x=100, y=50)
e2 = Entry(top, width=20, textvariable=b).
place(x=100, y=90)
xx = partial(xx, a, b)
submitbtn = Button(top, text="Submit", bg="white",
fg="blue", command=xx).place(x=100, y=120)

top.mainloop()
```

**107. Explain the function of the tell() method.****Ans.**

- It tells the current position within the file.
- This indicates that the next read or write will occur at that many bytes from the beginning of the file.

**108. Explain the function of the seek() method.****Ans.** Move the current file position to a different location at a defined offset.**109. What will be the output of the following code?****Ans.**

```
fo = open("myfile.txt", "w+")
print ("Name of the file: ", fo.name)
```

```
# Assuming that the file contains these lines
# PythonQuestionnair
# Hello Viewers!!
```

```
seq=" PythonQuestionnair \nHello Viewers!!"
fo.writelines(seq )
fo.seek(0,0)

for line in fo:
print (line)

fo.close()
```

The output will be as follows:

Name of the file: myfile.txt

PythonQuestionnair

Hello Viewers!!

### 110. What is pickling in Python?

**Ans.**

- It is a process to convert a Python object into a byte stream.
- It is done using two methods dump and load.
- Serialization is an alternate name for pickling.

### 111. Guess output of the following.

```
with open("hello.txt", "w") as f:
    f.write("Good Morning Students")
```

```
with open('hello.txt', 'r') as f:
    data = f.readlines()
    for line in data:
        words = line.split()
    print (words)
f.close()
```

**Ans.** The output is as follows:

['Good', 'Morning', 'Students']

- 112. Create a table “friends” with “name, phone” column and enter value in it.**

**Ans.**

```
import mysql.connector

# database name is "db" contains "friends" table

mydb = mysql.connector.connect(
host="localhost",
user="root",
password="computers",
database="db"
)

mycursor = mydb.cursor()
mycursor.execute("create table friends(fname
varchar(20),phone Bigint(10))")

n = input("Enter name of friend")
p = int(input("phone no"))
sql = "insert into friends(fname,phone)
values(%s,%s)"
val = (n, p)

mycursor.execute(sql,val)
mydb.commit()

mydb.close()
```

**Table:**

| Fname | Phone      |
|-------|------------|
| Swati | 1234567899 |

**Table 8**

- 113. In the preceding example, add another column birthday after the phone column.**

**Ans.**

```
import mysql.connector
```

```

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="computers",
  database="db"
)

mycursor = mydb.cursor()
mycursor.execute("alter table friends add column
bdy date")

mydb.commit()
mydb.close()

```

| Fname | Phone      | Bday |
|-------|------------|------|
| Swati | 1234567899 |      |

**Table 9****114. Update the preceding table and set Bday of a friend****. Ans.**

```

import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="computers",
  database="db"
)

mycursor = mydb.cursor()

n = input("Enter nameof friend")
dt: str = input("Enter date of birth")

sql = "update friends set bdy=%s where fname=%s"
val = (dt,n)
mycursor.execute(sql, val)

```

```
mydb.commit()  
mydb.close()  
  
# Create a database and table and run above code for  
output.
```

**115. Delete the desired record from the preceding table.****Ans.**

```
import mysql.connector  
  
mydb = mysql.connector.connect(  
host="localhost",  
user="root",  
password="computers ",  
database="db"  
)  
  
mycursor = mydb.cursor()  
  
n = input("Enter name of friend")  
sql = "delete from friends where fname=%s"  
val = ( n, )  
  
mydb.commit()  
mydb.close()  
  
# Create a database and table and run above code for  
output.
```

**116. Create table users in MySQL with uname and pwd fields.**

**Create a login box in Python GUI, user may signup, signin,  
change password, delete account on button click.**

**Ans.**

```
from functools import partial  
from tkinter import *  
from tkinter import messagebox  
  
import mysql.connector  
  
mydb = mysql.connector.connect(
```

```
host="localhost",
user="root",
password="computers",
database="db"
)
mycursor = mydb.cursor()

top = Tk()

top.title("Login Box")
top.geometry("400x300+200+200")
top.configure(bg='pink')

def ins(un, pw):
    a = un.get()
    b = pw.get()

    sql = "insert into users(uname,pwd) values(%s,%s)"
    val = (a, b)

    mycursor.execute(sql, val)
    messagebox.showinfo("Message", 'Account Created')
    mydb.commit()

    un.set("")
    pw.set("")

def login(un,pw):
    a = un.get()
    b = pw.get()

    sql = "select * from users where uname=%s and
    pwd=%s"
    val = (a, b)
    mycursor.execute(sql, val)

    result=mycursor.fetchall()
```

```
if result:  
    messagebox.showinfo("Message",'successfully logIn  
    into account')  
else:  
    messagebox.showerror("Message","Try Again!")  
  
un.set("")  
pw.set("")  
  
def dels(un,pw):  
    a = un.get()  
    b = pw.get()  
  
    sql = "delete from users where uname=%s and pwd=%s"  
    val = (a, b)  
    mycursor.execute(sql, val)  
  
    messagebox.showinfo("Message",'Account Deleted')  
  
    mydb.commit()  
    un.set("")  
    pw.set("")  
  
def chngP(un, pw,cpw,npw):  
    a = un.get()  
    b = pw.get()  
  
    sql = "update users set pwd=%s where uname=%s and  
    pwd=%s"  
    val = (cpw.get() , a, b)  
  
    if cpw.get() == npw.get() :  
        mycursor.execute(sql, val)  
        messagebox.showinfo("Message", 'Password Updated')  
        mydb.commit()  
  
    else:  
        messagebox.showerror("Message","password does not
```

```
match")  
  
un.set("")  
pw.set("")  
npw.set("")  
cpw.set("")  
  
# creating label  
  
uname = Label(top, text="Username", fg='blue',  
bg='pink').place(x=30, y=50)  
password = Label(top, text="Password", bg='pink',  
fg='blue').place(x=30, y=90)  
Confirm_password = Label(top, text="New Password",  
bg='pink', fg='blue').place(x=30, y=190)  
New_password = Label(top, text="Confirm Password",  
bg='pink', fg='blue').place(x=30, y=230)  
  
# creating text box  
un =StringVar()  
pw = StringVar()  
cpw = StringVar()  
npw = StringVar()  
  
e1 = Entry(top, width=30, textvariable=un).  
place(x=170, y=50)  
e2 = Entry(top, width=30, textvariable=pw).  
place(x=170, y=90)  
e3 = Entry(top, width=30, textvariable=npw).  
place(x=170, y=190)  
e4 = Entry(top, width=30, textvariable=cpw).  
place(x=170, y=230)  
  
ins=partial(ins,un, pw)  
login=partial(login,un,pw)  
dels=partial(dels,un,pw)  
chngP=partial(chngP,un,pw,npw,cpw)  
  
# creating button
```

```

insertbtn = Button(top, text="SignUp", bg="blue",
fg="pink", width="10", command=ins).place(x=50,
y=140)

signinbtn = Button(top, text="SignIn", bg="blue",
fg="pink", width="10", command=login).place(x=140,
y=140)

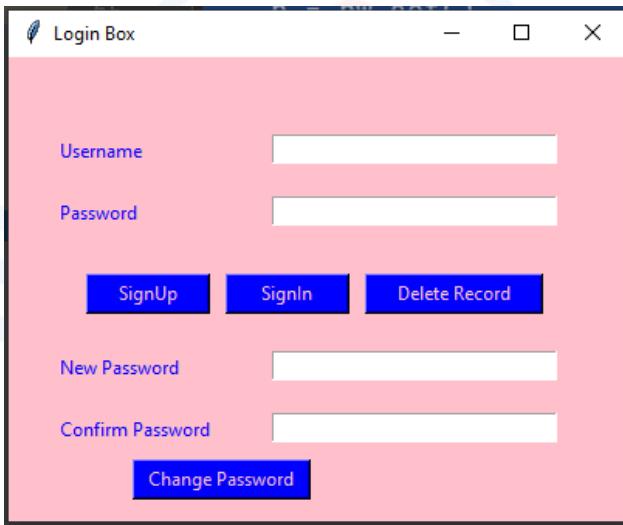
updatebtn = Button(top, text="Change Password",
bg="blue", fg="pink", width="15", command=chngP).
place(x=80, y=260)

deletebtn = Button(top, text="Delete Record",
bg="blue", fg="pink", width="15", command=dels).
place(x=230, y=140)

top.mainloop()

```

The following screen will be visible:



*Figure 33*

- 117. Create a GUI form to insert the employee name and generate employee ID.**

**Ans.**

```

from functools import partial from tkinter import *
import tkinter as tk
from tkinter import messagebox import mysql.connector

```

```
top = Tk()
top.title("Generate Id ")
top.geometry("300x150+200+200")
top.config(bg="lightgray")

mydb = mysql.connector.connect(
host="localhost",
user="root",
password="1234",
database="db"
)
mycursor = mydb.cursor()

def generate(name):
    a = IntVar

    mycursor.execute("select max(eid) from employee")

    result = mycursor.fetchone()
    a = result[0] + 1
    messagebox.showinfo("Id:",a)

    sql = "insert into employee(eid,name) values(%s,%s)"
    val = (a, name.get())
    mycursor.execute(sql, val)
    messagebox.showinfo("Message", 'Id Generated')
    mydb.commit()

    name = StringVar()

    nam = Label(top, text="Enter your Name..",
    bg="lightgray", fg="black").place(x=50, y=30)
    e1 = Entry(top, textvariable=name, width="30").
    place(x=70, y=50)
    #lid = Label(top, text=" Please Note your ID : ",
    fg='black',
    bg='lightgray').place(x=50, y=160)
    #answer = tk.Label(top, fg='black').place(x=200,
```

```
y=160)

generate = partial(generate, name)

buttl = Button(top, text="Generate ID", width="30",
bg="black", fg="white", command=generate).
place(x=40, y=100)

top.mainloop()
```

The following is the screenshot:



*Figure 34*

### **118. What is SQLite?**

**Ans.** Like other DBMS SQLite is an RDBMS and light-weighted. It is designed by *D.Richard Hipp* for the purpose of no administration required for operating a program. SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT). SQLite doesn't require configuration.

### **119. What is the difference between SQL and SQLite?**

**Ans.** The main differences between SQL and SQLite are:

- SQL is Structured Query Language while SQLite is a relational database management system mostly used in android mobile devices to store data.
- SQL support stored procedures while SQLite does not support stored procedures.
- SQL is server-based, while SQLite is file-based.

**120. What is the use of the LIMIT clause with the SELECT query?**

**Ans.** The LIMIT clause is used with the SELECT statement when we want a limited number of fetched records.

**Syntax:**

```
SELECT column1, column2, columnN
FROM table_name
LIMIT [no of rows]
```

**Example:**

```
SELECT * FROM STUDENT LIMIT 5; #it will return five records from student table
```

OFFSET is used in some cases where we have to retrieve the records starting from a certain point. Select 3 records form table "STUDENT" starting from 4th position.

```
SELECT * FROM STUDENT LIMIT 3 OFFSET 3;
```

**121. What is the difference between UNION and UNION ALL operator?**

**Ans.** UNION ALL     The operator is used to combine the result of two or more tables using the SELECT statement. The UNION operator ignores the duplicate entries while combining the results, while UNION ALL doesn't ignore duplicate values.

**Syntax:**

```
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
UNION ALL
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];
```

**122. What is SQLite JOIN? How many types of JOINS are supported in SQLite?**

**Ans.** JOIN The SQLite clause is used to combine two or more tables in a database. It combines the table by using the

common values of both tables. There are mainly three types of joins supported in SQLite:

- **SQLite INNER JOIN:** SQLite INNER JOIN is the simplest and most common join. It combines all rows from both tables where the condition is satisfied.

**Syntax:**

```
SELECT ... FROM table1 [INNER] JOIN table2 ON  
conditional_expression ...
```

**Example:**

```
SELECT EMP_ID, NAME, DEPT FROM EMP  
INNER JOIN INCR  
ON EMP.ID = INCR.EMP_ID;
```

- **SQLite OUTER JOIN:** The SQLite left outer join is used to fetch all rows from the left-hand table specified in the ON condition and only those rows from the right table where the join condition is satisfied.

**Syntax:**

```
SELECT ... FROM table1 LEFT OUTER JOIN table2  
ON conditional_expression
```

**Example:**

```
SELECT EMP_ID, NAME, DEPT FROM EMP LEFT  
OUTER JOIN INCR  
ON EMP.ID = INCR.EMP_ID;
```

- **SQLite CROSS JOIN:** The SQLite Cross join is used to match every rows of the first table with every rows of the second table.

**Syntax:**

```
SELECT ... FROM table1 CROSS JOIN table2
```

**Example:**

```
SELECT * FROM TEAM1 CROSS JOIN TEAM2;
```

### 123. What is the SQLite Julian day function?

**Ans.** A Julian Day is the number of days since Nov 24, 4714 BC 12:00pm Greenwich time in the Gregorian calendar. The julianday function returns the date as a floating-point number.

**Example:**

```
SELECT julianday('2020-01-10 10:55:20');
SELECT julianday('now');
```

**124. What is Primary key and Foreign Key in SQLite?**

**Ans.** The SQLite primary key is a simple field or a combination of fields that are used to uniquely define a record. A table can have only one primary key. A primary key should not be a NULL value. SQLite Foreign Key is used to specify that values in one table also appear in another table. It enforces referential integrity within the SQLite database. The referenced table is known as the parent table while the table with the foreign key is known as the child table. The foreign key in the child table will generally reference a primary key in the parent table.

**125. What is the NoSQL database?**

**Ans.** NoSQL stands for “Not Only SQL”. NoSQL is a type of database that can handle and sort all types of unstructured, messy, and complicated data. It is just a new way to think about the database. NoSQL is an open-source database. It has no approach for the backup of data. It has some negative points such as no GUI, no Backup, Open source, and has no reliable standard.

**126. What is Redis?**

**Ans.** Redis is a NoSQL database that follows the principle of key-value stores. Redis is a NoSQL database so it facilitates users to store huge amounts of data without the limitations of a Relational database. Redis supports various types of data structures such as strings, hashes, lists, sets, sorted sets, bitmaps, hyperlogs, and geospatial indexes with radius queries.

**127. What is the difference between Redis and RDBMS?**

**Ans.** The following table shows the difference:

| Redis                                                        | RDBMS                              |
|--------------------------------------------------------------|------------------------------------|
| Redis stores everything in primary memory. secondary memory. | RDBMS stores everything in memory. |

|                                                                                                                                                                                        |                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| In Redis, Read and Write operations are extremely fast because of operations are slow because of storing data in primary memory.                                                       | In RDBMS, Read and Write are slow because of operations are slow because of storing data in secondary memory.                        |
| Primary memory is lesser in size and much expensive than secondary. Therefore, Redis cannot store large files or binary data.                                                          | Secondary memory is abundant in size and is cheaper than primary memory. Therefore, RDBMS can easily deal with these types of files. |
| Redis is used only to store small textual information that has less frequent usage and not needs to be accessed, modified, required to be very fast. and inserted at a very fast rate. | RDBMS can hold large data that                                                                                                       |
| If you try to write bulk data more than the available memory, then you will receive errors.                                                                                            |                                                                                                                                      |

**Table 10**

### 128. What is the difference between MongoDB and Redis database?

**Ans.** The following is the difference:

| Comparison Index | Redis                                                                                      | MongoDB                                                                                       |
|------------------|--------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| Introduction     | Redis is an in-memory data structure store, used as a database, cache, and message broker. | MongoDB is one of the most popular NoSQL databases that follow the document stores structure. |
| Primary database | Redis follows the key-value store model.                                                   | MongoDB follows the document store model.                                                     |
| Official Website | <a href="http://redis.io">redis.io</a>                                                     | <a href="http://www.mongodb.com">www.mongodb.com</a>                                          |
| Developed By     | Redis is developed by Salvatore Sanfilippo.                                                | MongoDB is developed by MongoDB Inc.                                                          |
| Initial Release  | Redis is initially released in 2009.                                                       | MongoDB is also initially released in 2009.                                                   |
| Cloud-based      | No                                                                                         | No                                                                                            |
| Implementation   | Redis is written and Language implemented in the C language.                               | MongoDB is written and implemented in the C++ language.                                       |

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |  |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Server operating systems        | BSD, Linux, OS X, Linux, OS X, Solaris, Windows<br>Windows                                                                                                                                                                                                                                                                                                                                                                                       |  |
| Data Scheme                     | schema-free schema-free                                                                                                                                                                                                                                                                                                                                                                                                                          |  |
| Secondary Indexes               | No Yes                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |
| SQL                             | No No                                                                                                                                                                                                                                                                                                                                                                                                                                            |  |
| APIs and other access methods   | Redis follows a proprietary protocol. MongoDB follows a proprietary protocol using JSON.                                                                                                                                                                                                                                                                                                                                                         |  |
| Supported programming languages | C, C#, C++, Clojure, Actionscript, C, C#, C++, Crystal, D, Dart, Elixir, Clojure, ColdFusion, D, Erlang, Fancy, Go, Dart, Delphi, Erlang, Haskell, Haxe, Java, Go, Groovy, Haskell, JavaScript (Node.js), Java, JavaScript, Lisp, Lisp, Lua, MatLab, Lua, MatLab Perl, PHP, Objective-C, OCaml, PowerShell, Prolog, Perl, PHP, Prolog, Pure Python, R, Ruby, Scala, Data, Python, R, Rebol, Smalltalk, Ruby, Rust, Scala, Scheme, Smalltalk, Tcl |  |
| Server-side scripts             | Lua JavaScript                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |
| Triggers                        | No No                                                                                                                                                                                                                                                                                                                                                                                                                                            |  |
| Partitioning methods            | Redis uses Sharding for partition. MongoDB also uses Sharding for partition. Redis follows master-                                                                                                                                                                                                                                                                                                                                               |  |
| Replication methods             | MongoDB also follows slave replication. master-slave replication.                                                                                                                                                                                                                                                                                                                                                                                |  |
| MapReduce                       | No Yes                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |
| Consistency concepts            | Eventual Consistency and Immediate Consistency                                                                                                                                                                                                                                                                                                                                                                                                   |  |
| Foreign keys                    | No No                                                                                                                                                                                                                                                                                                                                                                                                                                            |  |
| Transaction concepts            | Optimistic locking, No atomic execution of commands blocks, and scripts.<br>Yes Yes                                                                                                                                                                                                                                                                                                                                                              |  |
| Concurrency                     | No Yes                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |
| MapReduce                       | Yes Yes                                                                                                                                                                                                                                                                                                                                                                                                                                          |  |
| Durability                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |  |

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                         |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| In-memory capabilities  | Yes Yes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                         |
| User concepts           | Simple password-based access control. and roles                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Access rights for users |
| Special Characteristics | Redis is ranked as MongoDB is considered the world's fastest as the next-generation database. It reduces database application complexity, Helped many businesses simplifies development, to transform their accelerates time to industries by providing market, and provides big data. The world's unprecedented most sophisticated flexibility to developers organizations, from with its visionary data cutting-edge startups to structures and modules. the largest companies, use MongoDB to create applications never before possible, at a very low cost.                                                                                                                                                                                                                                                                                                                                                                                          |                         |
| Comparing Advantages    | Redis is an in-memory MongoDB provides database platform the best of traditional that provides support databases as well as of wide range of flexibility, scale, and data structures such performance required as strings, hashes, by today's applications. sets, lists, sorted sets, MongoDB is a database bitmaps, hyperloglogs, of giant ideas. MongoDB and geospatial indexes. keeps the most valuable Redis provides features of the Relational effortless scaling in database, that is, strong a fully automated consistency, expressive manner by overseeing query language, and all the operations of secondary indexes. It sharding, re-sharding, facilitates developers to and migration. It also build highly functional includes persistence, applications faster than instant automatic failure detection, backup NoSQL databases. and recovery, and in- memory replication across racks, zones, datacenters, regions, and cloud platforms. |                         |

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Key Customers | Key customers of Redis are Verizon, Vodafone, MongoDB are: ADP, Atlassian, Trip Advisor, Adobe, AstraZeneca, Jet.com, Nokia, BBVA, Bosch, Cisco, Samsung, HTC, Docker, CERN, Department Staples, Intuit, Groupon, of Veteran Affairs, Shutterfly, KPMG, TD, eBay, eHarmony, Bank, UnitedHealthcare, Electronic Arts, Expedia, RingCentral, The Facebook's Parse, Motley Fool, Bleacher Report, Forbes, Foursquare, HipChat, Genentech, MetLife, Salesforce, Hotel Pearson, Sage, Tonight, Cirruspath, Salesforce, The Weather Itslearning.com, Xignite, Channel, Ticketmaster, Chargify, Rumble Under Armour, Verizon Entertainment, Scopely, Havas Digital, Revmob, MSN, Bleacher Report, Mobli, TMZ, Klarna, Shopify, etc. | Key customers of MongoDB are: ADP, Atlassian, Trip Advisor, Adobe, AstraZeneca, Jet.com, Nokia, BBVA, Bosch, Cisco, Samsung, HTC, Docker, CERN, Department Staples, Intuit, Groupon, of Veteran Affairs, Shutterfly, KPMG, TD, eBay, eHarmony, Bank, UnitedHealthcare, Electronic Arts, Expedia, RingCentral, The Facebook's Parse, Motley Fool, Bleacher Report, Forbes, Foursquare, HipChat, Genentech, MetLife, Salesforce, Hotel Pearson, Sage, Tonight, Cirruspath, Salesforce, The Weather Itslearning.com, Xignite, Channel, Ticketmaster, Chargify, Rumble Under Armour, Verizon Entertainment, Scopely, Havas Digital, Revmob, MSN, Bleacher Report, Mobli, TMZ, Klarna, Shopify, etc. |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Table 11****129. In which language MongoDB is written?**

**Ans.** MongoDB is written and implemented in C++.

**130. Does MongoDB database have tables for storing records?**

**Ans.** No. Instead of tables, MongoDB uses “Collections” to store data.

**131. What is sharding in MongoDB?**

**Ans.** In MongoDB, Sharding is a procedure of storing data records across multiple machines. It is a MongoDB approach to meet the demands of data growth. It creates a horizontal partition of data in a database or search engine. Each partition is referred to as a shard or database shard.

**132. Which are the different languages supported by MongoDB?**

**Ans.** MongoDB provides official driver support for C, C++, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go, and Erlang.

**133. What type of DBMS is MongoDB?**

**Ans.** MongoDB is a document-oriented DBMS

**134. Does MongoDB support primary-key, foreign-key relationships?**

**Ans.** No. By default, MongoDB doesn't support the primary key-foreign key relationship.

**135. Is there any need to create database command in MongoDB? Ans.**

You don't need to create a database manually in MongoDB because it creates automatically when you save the value into the defined collection for the first time.

**136. In which language is Redis written?**

**Ans.** Redis is written in ANSI C and mostly used for cache solution and session management. It creates unique keys for store values.

**137. What are the different languages supported by Redis?**

**Ans.** Redis supports a variety of languages, that is, C, C++, C#, Ruby, Python, Twisted Python, PHP, Erlang, Tcl, Perl, Lua, Java, Scala, and so on. If your favorite language is not supported yet, you can write to your client library, as the Protocol is pretty simple.

**138. What is NumPy?**

**Ans.** NumPy stands for numeric Python, which is a Python package for the computation and processing of the multidimensional and single-dimensional array elements. You can install NumPy by the following command:  
Python –m pip install numpy

**139. What is Ndarray?**

**Ans.** Ndarray means an-dimensional array object defined in NumPy. It stores collection of similar datatypes.

**Syntax:**

```
numpy.array(object, dtype = None, copy = True, order =  
None, subok = False, ndmin = 0)
```

Take a look at the following table:

| Parameter | Description                                                                                                                                                                 |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Object    | It represents the collection object. It can be a list, tuple, dictionary, set, and so on.                                                                                   |
| Dtype     | We can change the data type of the array elements by changing this option to the specified type. The default is none.                                                       |
| Copy      | It is optional. By default, it is true which means the object                                                                                                               |
| Order     | is copied.<br>There can be three possible values assigned to this option.                                                                                                   |
| Subok     | It can be C (column order), R (row order), or A (any).<br>The returned array will be base class array by default. We can change this to make the subclasses pass through by |
| Ndmin     | setting this option to true.<br>It represents the minimum dimensions of the resultant array.                                                                                |

**Table 12**

#### 140. Give an example of Ndarray in numPy.

**Ans.**

```
import numpy as np

a = np.array([[10,20],[30,40],[50,60],[70,80]])
print("printing the original array..")
print(a)

a=a.reshape(2,4)
print("printing the reshaped array..")
print(a)
```

#### 141. Create a numpy array using an existing tuple and list.

**Ans.**

```
import numpy as np
a=(1,2,3,4)
b=[10,20,30]
c=[a,b]
d=np.array(c)
print(d)
```

**142. Print a transpose numpy array. Ans.**

```
import numpy as np  
  
b=[[1,2,3],[10,20,30],[100,200,300]]  
  
d=np.array(b)  
print(d)  
  
e=d.T  
print(e)
```

**143. Write a program to display the output as desired (Find output)**

```
import numpy as np  
c=np.array([[1,2,3,4,5],[6,7,8,9,10]])  
  
find output : 3,5,8,10
```

**Ans. Solution:**

```
print(c[[0,0,1,1],[2,4,2,4]])  
or  
print(c[:,2::2])
```

**144. What is broadcasting? Explain with example?**

**Ans.** When the shapes of the two arrays are not similar and therefore they cannot be multiplied together, NumPy can perform such operations by using the concept of broadcasting. Broadcasting can be applied to the arrays if the following rules are satisfied:

- All the input arrays have the same shape.
  - Arrays have the same number of dimensions, and the length of each dimension is either a common length or 1.
  - Array with the fewer dimension can be appended with '1' in its shape.

**Example:**

```

import numpy as np

a = np.array([[1,2,3,4],[2,4,5,6],[10,20,39,3]])
b = np.array([2,4,6,8])
print("\nprinting array a..")
print(a)

print("\nprinting array b..")
print(b)
print("\nAdding arrays a and b ..")
c = a + b;
print(c)

print("\nMultiplying arrays a and b ..")
d=a*b
print(d)

```

The following image explains the preceding code in detail:

| a (3 X 4) |    |    |   | b (4) |   |   |   | stretch | = |    |    |    |    |
|-----------|----|----|---|-------|---|---|---|---------|---|----|----|----|----|
| 1         | 2  | 3  | 4 | 2     | 4 | 6 | 8 |         |   | 3  | 6  | 9  | 8  |
| 2         | 4  | 5  | 6 | 2     | 4 | 6 | 8 |         |   | 4  | 8  | 11 | 14 |
| 10        | 20 | 39 | 3 | 2     | 4 | 6 | 8 |         |   | 12 | 24 | 45 | 11 |

a = [[1, 2, 3, 4], [2, 4, 5, 6], [10, 20, 39, 3]]

b = [[2, 4, 6, 8]]

Figure 35

- 145. Write a program create a numpy array using list of numbers. Display the values less than 80 otherwise 0 from numpy array?**

```

import numpy as np

a = np.array([[101,28,86,54],[22,74,95,46],
[10,20,239,37]])

b=np.where(a<80,a,0)
print(b)

```

**146. Sort array by the given age or height?****Ans.**

```
import numpy as np

dtype = [('name', 'S10'), ('height', float), ('age', int), ('gender', 'S10')]
values = [('Swati', 5.4, 28, 'F'), ('Aavya', 4.2, 19, 'F'), ('Nidhi', 5.8, 37, 'F'), ('Deep', 6.1, 46, 'M')]

x=np.array(values, dtype=dtype)
print(x)

y=np.sort(x, order='age')
print(y)

z=np.sort(x, order=['age','height'])
print(z)
```

`dtype ()` is used to create datatype object. The output screenshot is as follows:

---

```
[(b'Swati', 5.4, 28, b'F') (b'Aavya', 4.2, 19, b'F')
 (b'Nidhi', 5.8, 37, b'F') (b'Deep', 6.1, 46, b'M')]
 [(b'Aavya', 4.2, 19, b'F') (b'Swati', 5.4, 28, b'F')
 (b'Nidhi', 5.8, 37, b'F') (b'Deep', 6.1, 46, b'M')]
 [(b'Aavya', 4.2, 19, b'F') (b'Swati', 5.4, 28, b'F')
 (b'Nidhi', 5.8, 37, b'F') (b'Deep', 6.1, 46, b'M')]
```

---

*Figure 36*

**147. Explain datatype object (dtype)**

- . **Ans**datatype object (an instance of `numpy.dtype` class) describes how the bytes in the fixed-size block of memory corresponding to an array item should be interpreted. It describes the following aspects of the data:
- Type of the data (integer, float, Python object, to name a few)

- Size of the data (how many bytes it is in, for example, the integer)
- The byte order of the data (little-endian or big-endian)
- If the data type is a record, an aggregate of other data types, (for example, describing an array item consisting of an integer and a float) :
  - What are the names of the “fields” of the record, by which they can be accessed?
  - What is the data-type of each field?
  - Which part of the memory block each field takes?

#### **148. Explain Pandas?**

**Ans.** Pandas is a high-level data manipulation tool developed by ~~Wes McKinney~~ on the Numpy package and its key data structure is called DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables. It is defined as an open-source library that provides high-performance data manipulation in Python. The name of Pandas is derived from the word Panel Data, which means an Econometrics from Multidimensional data.

Before Pandas, Python was capable of data preparation, but

it only provided limited support for data analysis. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, for example, load, manipulate, prepare, model, and analyze. The Pandas provides two data structures for processing the data, that is, Series and DataFrame:

- **Series:** It is defined as a one-dimensional array that is capable of storing various datatypes. The row labels of series are called the index.
- **Dataframe:** It is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data and has two different indexes, that is, row index and column index.

**149. Create Series and also change its index in Python pandas. Ans.**

```
import pandas as pd  
import numpy as np  
  
name = np.array(['s','w','a','t','i'])  
a = pd.Series(name)  
print(a)  
  
a.index=["1.1","1.2","1.3","1.4","1.5"]  
print(a)
```

**150. Create DataFrame from dictionary and also change its index in Python pandas.**

**Ans.**

```
import pandas as pd  
  
dict = {"country": ["India", "Russia", "Australia",  
"United Kingdom", "South Africa"],  
"area": [9.516, 17.10, 3.286, 9.597,  
12.221],  
"population": [200.4, 143.5, 1252, 1357,  
52.98] }  
  
a = pd.DataFrame(dict)  
print(a)  
  
a.index=["IN","RU","AUS","UK","SA"]  
print(a)
```

**151. How to read a CSV file and perform an operation?**

**Emp.CSV:**

**Empid,EName,Job,Salary,Deptno**

**101,Jerry,manager,456784,10**

**102,Smith,clerk,765432,20**

**103,Mike,admin,235678,10**

**104,Scott,manager,435678,10**

**105,James,clerk,65436****7,20****106,Bill,clerk,347623,10..... Ans.**

Reading the preceding CSV file:

```

import pandas

df=pandas.read_csv('emp.csv')
print(df)

print("\nCount total employees")
print("Employees in company =",df['Empid'].count())
print("\nFind out maximum salary")
print("maximum salary=",df['Salary'].max())
print("\nDisplay mean ,std,max,min")
print(df.describe())
print("\nPrint departmentwise total salary")
dg=df.groupby('Deptno')
print(dg['Salary'].sum())

print("OR")
print(df.groupby('Deptno')['Salary'].sum())

print("\nPrint Name of all clerks")
print(df[df['Job']=='clerk']['EName'])
s=df['Salary'].max()

print("\nName of the employee getting highest
salary")
print(df[df['Salary']==s]['EName'])

```

- 152. From the preceding CSV file, find the name and department number of employees getting salary more than 50000 and less than 800000?**

**Ans.**

```
import pandas
```

```
df=pandas.read_csv('emp.csv')

print(df[((df['Salary']>500000)&(df['Salary']<800000))][['EName','Deptno']]
```

### 153. Read the following excel file and perform operations:

Mobile.xlsx

|   | A       | B          | C     | D     |
|---|---------|------------|-------|-------|
| 1 | Company | Model      | Color | Price |
| 2 | Nokia   | 6.2        | black | 34566 |
| 3 | Samsung | Galaxy S20 | gray  | 97654 |
| 4 | Nokia   | 220 4G     | blue  | 12345 |
| 5 | Xolo    | ERA        | black | 21345 |
| 6 | Nokia   | 7.2        | white | 65473 |
| 7 | Redmi   | Note 9 Pro | gray  | 45678 |
| 8 | Redmi   | Note 9 Pro | white | 73455 |

Figure 37

**Sheet              Name:**

**Detail Ans.**

```
import pandas

df = pandas.read_excel('mobile.xlsx', sheet_name='Detail')
print(df)

print("\nColumn Name:\n")
print(df.columns)

print("\nfirst three records\n")
print(df.head(3))
print("\nlast two records\n")
print(df.tail(2))
print("\nExcel to CSV\n", df.to_csv())

print("\nFind Mobiles with price less than 60000\n")
print(df[df['Price']<60000][['Company','Model']])
```

**154. Create pivot table from excel data using pandas****? Ans.**

```
import pandas

df = pandas.read_excel('mobile.xlsx', sheet_name='Detail')
subset=df[['Company','Model','Price']]
print(subset)
print(subset.pivot_table(index='Company'))
```

**155. What kind of plots can you create in pandas?****Ans.**

- 'line': line plot (default)
- 'bar': vertical bar plot
- 'barh': horizontal bar plot
- 'hist': histogram
- 'box': boxplot
- 'kde': Kernel Density Estimation plot
- 'density': same as 'kde'
- 'area': area plot
- 'pie': pie plot
- 'scatter': scatter plot
- 'hexbin': hexbin plot

**156. Create a scatter and bar****plot? Ans.**

```
import matplotlib.pyplot as plt
import pandas as pd

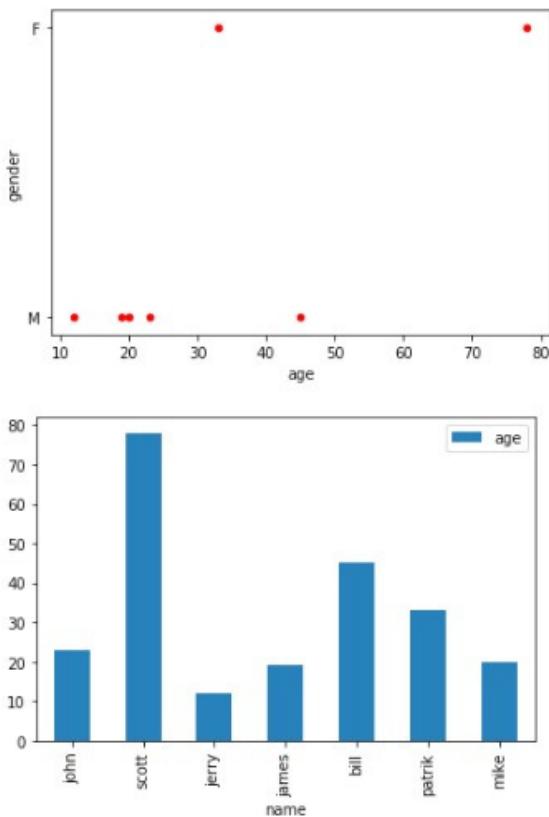
df = pd.DataFrame({
    'name':['john','scott','jerry','james',
    'bill','patrik','mike'],
    'age':[23,78,12,19,45,33,20],
    'gender':['M','F','M','M','M','F','M']
})
# a scatter plot
```

```
df.plot(kind='scatter',x='age',y='gender',color= 'red')
plt.show()

# a bar plot

df.plot(kind='bar',x='name',y='age')
plt.show()
```

The following is the output:



*Figure 38*

### 157. Explain syntax of plot in pandas.

**Ans.**

```
DataFrame.plot(x=None, y=None, kind='line',
ax=None, subplots=False, sharex=None, sharey=False,
```

```
layout=None, figsize=None, use_index=True,
title=None, grid=None, legend=True, style=None,
logx=False, logy=False, loglog=False, xticks=None,
yticks=None, xlim=None, ylim=None, rot=None,
fontsize=None, colormap=None, table=False,
yerr=None, xerr=None, secondary_y=False, sort_
columns=False, **kwds)
```

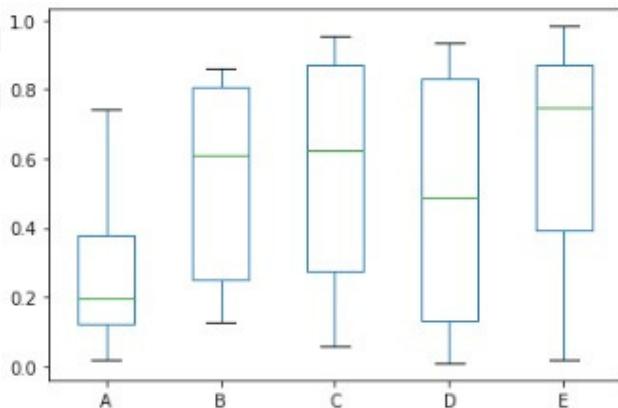
**158. Create a box****plot. Ans.**

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10, 5),
columns =['A', 'B', 'C', 'D', 'E'])

df.plot.box()
plt.show()
```

We get the following output:

**Figure 39****159. Create a scatter plot from columns of the CSV file****. Ans.**

```
import pandas
import matplotlib.pyplot as plt
```

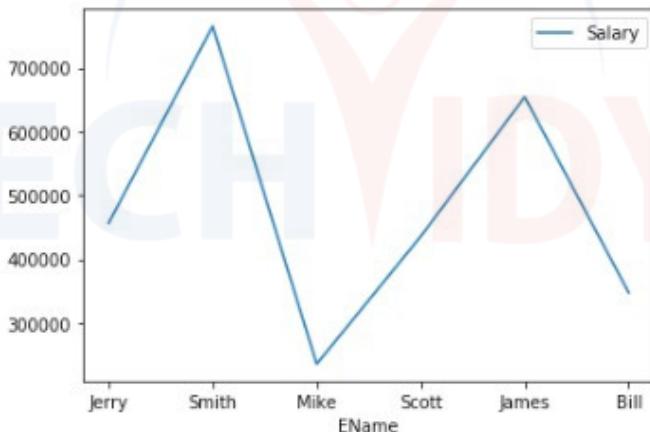
```
df=pandas.read_csv('emp.csv')

x=df['EName']
y=df['Salary']
plt.scatter(x,y)
plt.show()
```

**160. Create line plot on employee name and salary. Ans.**

```
import pandas
import matplotlib.pyplot as plt
df=pandas.read_csv('emp.csv')
df.plot.line(x='EName',y='Salary')
plt.show()
```

Refer to the following output:



*Figure 40*

**161. What is the difference between loc and iloc in Pandas? Give an example.**

**Ans.** Using these, we can practically do any data selection task on Pandas dataframes. loc is label-based, which means that we have to specify the name of the rows and columns that we need to filter out. On the other hand, iloc is integer index-based. So here, we have to specify rows and columns by their integer index:

```

import pandas as pd
import numpy as np

# create a sample dataframe

data = pd.DataFrame({
    'age' : [ 10, 22, 13, 21, 12, 11, 17],
    'section' : [ 'A', 'B', 'C', 'B', 'B', 'A', 'A'],
    'city' : [ 'Gurgaon', 'Delhi', 'Mumbai',
               'Delhi', 'Mumbai', 'Delhi', 'Mumbai'],
    'gender' : [ 'M', 'F', 'F', 'M', 'M', 'M', 'F']
})

print(data.loc[data.age >= 15])

print(data.loc[(data.age >= 12) & (data.gender == 'M')])

print(data.loc[1:3])

print("\n using iloc\n")

print(data.iloc[[0,2]])

print(data.iloc[[0,2],[1,3]])

```

The output will be as follows:

|   | age | section | city   | gender |
|---|-----|---------|--------|--------|
| 1 | 22  | B       | Delhi  | F      |
| 3 | 21  | B       | Delhi  | M      |
| 6 | 17  | A       | Mumbai | F      |

|   | age | section | city   | gender |
|---|-----|---------|--------|--------|
| 3 | 21  | B       | Delhi  | M      |
| 4 | 12  | B       | Mumbai | M      |

|   | age | section | city   | gender |
|---|-----|---------|--------|--------|
| 1 | 22  | B       | Delhi  | F      |
| 2 | 13  | C       | Mumbai | F      |
| 3 | 21  | B       | Delhi  | M      |

using iloc

|   | age | section | city    | gender |
|---|-----|---------|---------|--------|
| 0 | 10  | A       | Gurgaon | M      |
| 2 | 13  | C       | Mumbai  | F      |

|   | section | gender |
|---|---------|--------|
| 0 | A       | M      |
| 2 | C       | F      |

**Figure 41**

**162. Explain the difference between Pandas and NumPy.**

**Ans.** The following table shows the difference:

| Basis for comparison | Pandas                                                                            | NumPy                                                      |
|----------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------|
| Works with           | Pandas module works with the tabular data.                                        | NumPy module works with numerical data.                    |
| Powerful Tools       | Pandas have powerful tools like Series, tool DataFrame, and so on.                | NumPy has a powerful like Arrays.                          |
| Organizational usage | Pandas is used in popular organizations such as Instacart, SendGrid, and Sighten. | NumPy is used in popular organizations such as SweepSouth. |
| Performance          | Pandas have a better performance for 500K rows or more.                           | NumPy has a better performance for 50K rows or less.       |
| Memory Utilization   | Pandas consume large memory as compared to NumPy.                                 | NumPy consumes less memory as to Pandas.                   |

*Table 13*

**163. How to get the items of series A not present in series B?**

**Ans.**

```
import pandas as pd

p1 = pd.Series([2, 4, 5, 6, 8, 9, 10])
p2 = pd.Series([5, 8, 10, 12, 14, 16])

print(p1[~p1.isin(p2)] )
```

The output is as follows:

```
0    2
1    4
3    6
5    9
dtype: int64
```

*Figure 42*

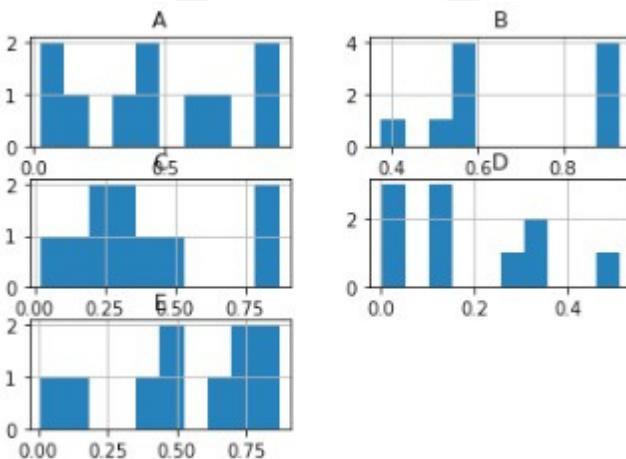
**164. Create a****histogram.** Ans.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10, 5),
columns=['A', 'B', 'C', 'D', 'E'])

df.hist()
plt.show()
```

The output will be as follows:



**Figure 43**

**165. What is a web framework?**

**Ans.** It is a code library that helps you to build and maintain a flexible and dynamic web application, web app, and web service. For example, Ruby on Rails for Ruby. Django is a web development framework for Python which offers a standard method for fast and effective website development.

**166. What is Django?**

**Ans.** Django is a web development framework written in Python. It helps you to build and maintain web applications.

It is based on the **MVT (Model View Template)** design pattern. This framework uses a famous tag line: the web framework for perfectionists with deadlines.

Install Django by pip:

```
pip install django == 2.0.3
```

### 167. Explain the MVT pattern.

**Ans. Model-View-Template (MVT)** The is a different concept compared to MVC. Django itself manages the Controller part. The Model helps to handle database. It is a data access layer that handles the data. The Template is a presentation layer (HTML file) that handles the user interface part completely.

The View is used to execute the business logic and interact with a model to carry data and renders a template. The following image explains the concept more clearly:

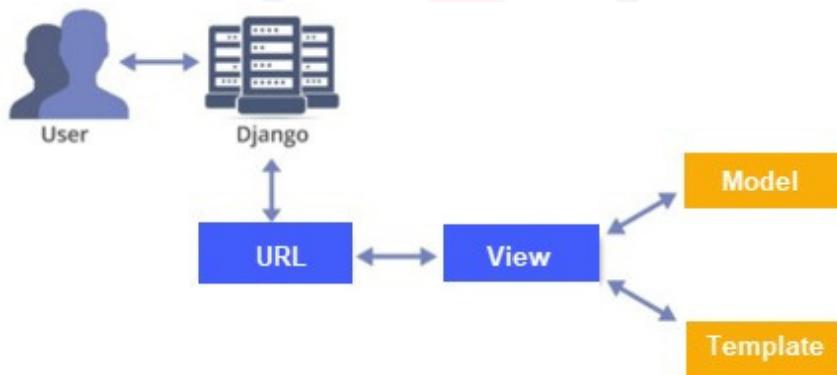


Figure 44

As seen in the preceding diagram, a user requests a resource for Django. Django acts as a controller and checks to the available resource in the URL. If URL maps, a view is called which interacts with model and template. Django then responds to the user and sends a template as a response.

### 168. How to check form validity in Django?

**Ans.** The `is_valid()` method is used to perform validation for each field of the form. It is defined in the Django Form class. It returns True if data is valid and place all data into a `cleaned_data` attribute.

**Example:**

```

def emp(request):
    if request.method == "POST":
        form = EmployeeForm(request.POST)
        if form.is_valid():
            try:
                return redirect('/')
            except:
                pass
        else:
            form = EmployeeForm()

```

**169. What is Django Rest Framework (DRF)?**

**Ans.** Django REST is a framework that lets you create RESTful APIs rapidly. RESTful APIs are perfect for web applications since they use low bandwidth and are designed such that they work well with the communications over the Internet such as GET, POST, and PUT.

**170. What are the various files that are created when you create a Django Project? Explain briefly.**

**Ans.** files with its description:

| File Name   | Description                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------|
| manage.py   | A command-line utility that allows you to interact with your Django project                         |
| __init__.py | An empty file that tells Python that the current directory should be considered as a Python package |
| settings.py | Consists of the settings for the current project                                                    |
| urls.py     | Contains the URLs for the current project                                                           |
| wsgi.py     | This is an entry-                                                                                   |

point for the web servers to serve

**Table 14**  
the project you have created

**171. What is Django ORM?**

**Ans.** Django ORM is one of the special feature-rich tools in Django. ORM is an acronym for Object-Relational Mapper.

Django ORM is the abstraction between models (web application data-structure) and the database where the data is stored. It makes it possible to retrieve, save, delete, and perform other operations over the database without ever writing any SQL code.

**172. How does Django compare to other popular frameworks such as Laravel?**

**Ans.** Django has many unique features. It allows for rapid development of applications without any security loopholes or performance lacking. Some more differences:

- Laravel is PHP based while Django is Python based, which is clearly more powerful and robust than PHP.
- The performance of Django is better than Laravel because of the different programming languages it uses.
- Django is based on the MTV architecture, a more robust and loosely coupled architecture while Laravel is strictly based on the MVC architecture.
- Django can use RegEx while Laravel doesn't provide you with that functionality.

**173. What is Jinja Templating?**

**Ans.** Django supports many popular templating engines and by default, it comes with one very powerful templating engine. Jinja Templating is a very popular templating engine for Python, the latest version in the market is Jinja 2.

**174. What is the difference between Flask and Django?**

**Ans.** The following table explains the difference:

| Comparison Factor        | Django                  | Flask                      |
|--------------------------|-------------------------|----------------------------|
| Project Type             | Supports large projects | Built for smaller projects |
| Templates, Admin and ORM | Built-in                | Requires installation      |

|                    |                                                                        |                                                                                                   |
|--------------------|------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| Ease of Learning   | Requires more learning and practice                                    | Easy to learn                                                                                     |
| Flexibility        | Allows complete web development without the need for third-party tools | More flexible as the user can select third-party tools according to their choice and requirements |
| Visual Debugging   | Does not support Visual Debug                                          | Supports Visual Debug                                                                             |
| Type of framework  | Batteries included                                                     | Simple, lightweight                                                                               |
| Bootstrapping-tool | Built-in                                                               | Not available                                                                                     |

**Table 15****175. What is the usage of Django-admin.py and manage.py?****Ans.**

- Django-admin.py: It is a Django's command-line utility for administrative tasks.
- Manage.py: It is an automatically created file in each Django project. It is a thin wrapper around the Django-admin.py. It has the following usage:
  - o It puts your project's package on sys.path.
  - o

It sets the DJANGO\_SETTING\_MODULE environment

**176. Is Django stable?** Django is quite stable. Many companies such as Instagram, Discus, Pinterest, and Mozilla have been using Django for many years now.

**177. Is Django a CMS?**

**Ans.** Django is not a CMS (content-management-system). It is just a web framework, a tool that allows you to build websites.

**178. Does the Django framework scale?**

**Ans.** Yes. Hardware is much cheaper when compared to the development time and this is why Django is designed to make

full use of any amount of hardware that you can provide it.

You can add hardware at any level, that is, database servers, caching servers, or web application servers.

### 179. What are the different types of Django Exception Classes?

Ans. `django.core.exceptions` module contains the following classes:

| Exception               | Description                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| AppRegistryNotReady     | It is raised when attempting to models before the app loading process.                                                            |
| ObjectDoesNotExist      | The base class for DoesNotExist exceptions.                                                                                       |
| EmptyResultSet          | If a query does not return any result, this exception is raised.                                                                  |
| FieldDoesNotExist       | It raises exception when the requested field does not exist.                                                                      |
| MultipleObjectsReturned | This exception is raised by a query if only one object is expected, but multiple objects are returned.                            |
| SuspiciousOperation     | This exception is raised when a user has performed an operation that should be considered suspicious from a security perspective. |
| PermissionDenied        | It is raised when a user does not have permission to perform the action requested.                                                |
| ViewDoesNotExist        | It is raised by <code>django.urls</code> when a requested view does not exist.                                                    |
| MiddlewareNotUsed       | It is raised when a middleware is not used in the server configuration.                                                           |
| ImproperlyConfigured    | The ImproperlyConfigured exception is raised when Django is somehow improperly configured.                                        |
| FieldError              | It is raised when there is a problem with a model field.                                                                          |
| ValidationError         | It is raised when data validation fails to form or model field validation                                                         |

**Table 16**

**180. Give an example to set and get a cookie.****Ans.**

```
def setcookie(request):
    response = HttpResponse("Cookie Set")
    response.set_cookie('python-tutorial', 'swati
computers')
    return response

def getcookie(request):
    tutorial = request.COOKIES['python-tutorial']
    return HttpResponse("python tutorials @: "+tutorial)
```

**181. Is it mandatory to use the model/ database layer in Django? **Ans.****

**Ans.** model/ database layer is actually decoupled from the rest of the framework.

**182. What is Python Flask?**

**Ans.** Flask is a web framework written in Python which provides libraries to build lightweight web applications. It is based on the WSGI toolkit and jinja2 template engine. Flask is considered as a micro framework.

**183. What is WSGI and Jinja template?**

**Ans.** WSGI stands for web server gateway interface which is a standard for Python web application development. Jinja2 is a web template engine that combines a template with a certain data source to render the dynamic web pages.

**184. What is the difference between Django, Pyramid, and Flask?**

**Ans.** Take a look at the following table for understanding the difference:

| Django                    | Pyramid                   | Flask                    |
|---------------------------|---------------------------|--------------------------|
| It is a Python framework. | It is the same as Django. | It is a micro framework. |

|                                         |                                              |                                          |
|-----------------------------------------|----------------------------------------------|------------------------------------------|
| It is used to build large applications. | It is the same as Django.                    | It is used to create small applications. |
| It includes an ORM.                     | It provides flexibility and the right tools. | It does not require external libraries.  |

**Table 17****185. What is Flask Sijax?**

external libraries.

**Ans.** Flask Sijax is a Simple Ajax and jQuery library. It is used to enable Ajax in web applications. It uses JSON to pass data between the server and the browser.

**186. Write a code to get querystring from Flask.****Ans.**

```
from flask import request
@app.route('/data')
def data () :
    model = request.args.get ('model_no')
```

**187. Explain cookies in Flask?**

**Ans.** Cookies are used to track the user's activities on the web and reflect some suggestions according to the user's choices to enhance the user's experience.

In a flask, the cookies are associated with the request

object

as the dictionary object of all the cookie variables and their values transmitted by the client. Flask facilitates us to specify

**Syntax:**

`response.setCookie(<title>, <content>, <expiry>)`  
the expiry time, path, and domain name of the website.

```
request.cookies.get(<title>)
```

**188. What is App routing and how to do it in Flask?**

**Ans.** App routing is used to map the specific URL with the associated function that is intended to perform some task. It is used to access some particular page:

```
@app.route('/home')
```

Flask facilitates us to add the variable part to the URL by using the section:

```
from flask import Flask
app = Flask(__name__)

@app.route('/home/<name>')
def home(name):
    return "hello,"+name;

if __name__ == "__main__":
    app.run(debug = True)
```

**189. Which method in Flask is used to give user feedback in a box like ‘alert’ in JavaScript and message’ in Tkinter?**

**Ans.** The `flash()` method is used to generate an informative message in Flask.

**190. What is containerization? What is docker?**

**Ans.** Docker is an open-source application that allows administrators to create, manage, deploy, and replicate applications using containers. Containers can be thought of as a package that houses dependencies that an application requires to run at an operating system level.

**191. What is web-scraping?**

**Ans.** If you need to pull a large amount of data from a website without going to that website, web scraping is used. It is used in the following situations: gathering e-mail ID to send bulk messages, pull out price details from an e-commerce site to compare, details regarding job openings, interviews are collected from different websites and then listed in one place so that it is easily accessible to the user.

Web scraping helps collect these unstructured data and store it in a structured form. There are different ways to scrape websites such as online Services, APIs, or writing your own code. To extract data using web scraping with Python, you need to follow these basic steps:

- i. Find the URL that you want to scrape
- ii. Inspecting the page

- iii. Find the data you want to extract
- iv. Write the code
- v. Run the code and extract the data
- vi. Store the data in the required format

To scrap data from Python, we have to install some libraries:

- requests
- html5lib
- bs4

You can install them from pip install in Python.

**BeautifulSoup:** Beautiful Soup is a Python package for parsing HTML and XML documents. It creates parse trees that are helpful to extract the data easily.

### Example

```
import requests  
import urllib.request  
from bs4 import BeautifulSoup  
  
url="https://www.monster.com/jobs/  
search/?q=Software-Developer&where=India"  
response= requests.get(url)  
  
#print(response)  
  
htmlcontent=response.content  
  
soup = BeautifulSoup(htmlcontent, "html.parser")  
  
#print(soup)  
  
#print(soup.prettify )  
  
print(soup.findAll('div'))
```

**192. Write a program to change the key value of the dictionary and change the attribute value.**

**Ans.**

```
vehicle={  
    "Tw": {"Name": "victor",  
           "Price": "70K",  
           "color": "red"},  
    "FW": {"Name": "kwid",  
           "Price": "70L",  
           "color": "blue"},  
}  
print(vehicle)  
  
vehicle["TV"] = vehicle.pop("Tw")  
print(vehicle)  
print(vehicle["TV"]["color"])  
vehicle["TV"]["color"] = "silver"  
print(vehicle)  
  
#Here I have created a dictionary Vehicle with key  
#“TW” and “FW”.  
#I changed the key “TW” by “TV”.  
#I have changed the “TV” color from red to silver
```

The following will be the output of the preceding code:

```
{'Tw': {'Name': 'victor', 'Price': '70K', 'color': 'red'}, 'FW': {'Name': 'kwid', 'Price': '70L', 'color': 'blue'}}  
{'FW': {'Name': 'kwid', 'Price': '70L', 'color': 'blue'}, 'TV': {'Name': 'victor', 'Price': '70K', 'color': 'red'}}  
{'FW': {'Name': 'kwid', 'Price': '70L', 'color': 'blue'}, 'TV': {'Name': 'victor', 'Price': '70K', 'color': 'silver'}}
```

**Figure 45**

**193. Write a program to create a nested list. Then, add new values and add a new list.**

**Ans.**

```
a=['a',[‘b’,’c’],’d’]
```

```
a[0]=10
print(a)
a[1][0]=20
print(a)
a[1][1]=['w']
a[1].append('pop')
print(a)
a[1][0]='q'
print(a)
```

The following will be the output of the above code:

---

```
[10, ['b', 'c'], 'd']
[10, [20, 'c'], 'd']
[10, [20, ['w']], 'pop'], 'd']
[10, ['q', ['w']], 'pop'], 'd']
```

---

*Figure 46*

#### 194. Create a set and perform a union, intersection, and difference operation.

**Ans.**

```
a={'pomogrenate','banana','guava','jamun','litchi'}
print(type(a))
for i in a:
    print(i)
a.add('kiwi')
a.add('grapes')
print(a)
a.remove("banana")
print(a)

b={'guava','banana','apple','chiku','cherry',
jamun'}
print("\n\n")
print("set A=",a)
```

```

print("set B=",b)
c=a.union(b)
print(c)

d=a.intersection(b)
print("\n",d)
i=a.difference(b)
print(i)
i=b.difference(a)
print(i)

```

The following will be the output of the preceding code:

```

<class 'set'>
guava
jamun
litchi
pomogrenate
banana
{'grapes', 'guava', 'kiwi', 'jamun', 'litchi', 'pomogrenate', 'banana'}
{'grapes', 'guava', 'kiwi', 'jamun', 'litchi', 'pomogrenate'}

set A= {'grapes', 'guava', 'kiwi', 'jamun', 'litchi', 'pomogrenate'}
set B= {'guava', 'jamun', 'chiku', 'apple', 'cherry', 'banana'}
{'grapes', 'guava', 'kiwi', 'jamun', 'chiku', 'pomogrenate', 'litchi', 'apple', 'cherry', 'banana'}
{'jamun', 'guava'}
{'pomogrenate', 'grapes', 'kiwi', 'litchi'}
{'apple', 'cherry', 'chiku', 'banana'}

```

**Figure 47**

### 195. Create a nested list and use extend and insert function

**s. Ans.**

```

n=[“1”, “2”, [“3”, “4”, ], “5”]      a=
[“a”, “b”]
print(n)
n.extend(a) n.insert(1,
(“q”, “p”)) print(n)

```

The following will be the output of the above code:

```

['1', '2', ['3', '4'], '5']
['1', ('q', 'p'), '2', ['3', '4'], '5', 'a', 'b']

```

**Figure 48**

**196. Write a program to count the occurrence of “scream” in the following sentence:**

**“I scream, you scream, all scream, for ice cream”?**

**Ans.**

```
n="I scream,you scream,all scream,for ice cream"  
print(n.count("scream"))
```

The following will be the output of the preceding code:

3

**197. Write a program to create a list of numbers and display the numbers divisible by 3.**

**Ans.**

```
n=[12,3,5,7,9,7,5,4,12]  
#1st method  
for i in range(0,len(n)):  
    if (n[i]%3==0):  
        print(n[i],end=",")  
  
print("\n\n")  
#2nd method  
for i in n:  
    if (i%3==0):  
        print(i,end=",")  
  
print("\b")
```

The following will be the output of the preceding code:

**12,3,9,12,**

**12,3,9,12**

---

**Figure 49**

**198. Write a program to enter your name and display every second letter.**

**Ans.**

```
n=input("Enter your name")
print(n[0:len(n):2])
```

The following will be the output of the preceding code:

---

```
Enter your nameswati computers jaipur
saicmuesjiu
```

---

*Figure 50*

**199. Write a program to enter your name and display it in the reverse order.**

**Ans.**

```
n=input("Enter your name")
print(n[::-1])
for i in range(len(n)-1,-1,-1):
    print(n[i],end="")
```

The following will be the output of the preceding code:

---

```
Enter your nameayush
hsuya
hsuya
```

---

*Figure 51*

**200. Create a list and perform remove( ), pop ( ), insert( ), and index( ).**

**Ans.**

```
list1=[1,4,3,4,6,2,3,4,8]
print(list1.count(4))
print(len(list1))

list1.remove(6)
print(list1)
```

```
print(list1.pop(2))
print(list1)

print("index of value 8 is : %d"%list1.index(8))

print(list1.insert(7,100))
print(list1)
```

The following will be the output of the preceding code:

```
3
9
[1, 4, 3, 4, 2, 3, 4, 8]
3
[1, 4, 4, 2, 3, 4, 8]
index of value 8 is : 6
None
[1, 4, 4, 2, 3, 4, 8, 100]
```

*Figure 52*

**201. Write a program to create a list and display the square of list elements.**

**Ans.**

```
a=[2,3,4]
b=[x*x for x in a]
print(b)
```

The following will be the output of the preceding code:

```
[4, 9, 16]
```

*Figure 53*

**202. Write a program to input string as “education” and display “edon.”**

**Ans.**

```
#education
#edon
```

```
name="Education"
print(name[0:2]+name[-2:])
```

The following will be the output of the preceding code:

Edon

**203. Create a tuple of five elements and unpack it in five different variables.**

**Ans.**

```
my_Tuple = (10, 20, 30, 40, 50)
a, b, c, d, e = my_Tuple
print(a)
print(b)
print(c)
print(d)
print(e)
```

The following will be the output of the preceding code:

10  
2  
0  
3  
0  
4  
0

**204. Write a program to create two tuples and swap them.**

**Ans.**

```
tuple1 = (10, 20)
tuple2 = (100, 200)
tuple1, tuple2 = tuple2, tuple1
print(tuple2)
print(tuple1)
```

The following will be the output of the preceding code:

(10,20)  
(100,200)

**205. Create a function fun( ) such that it can accept variable length of argument and print them.**

**Ans.**

```
def fun(*args):
    for i in args:
        print(i)
```

```
fun(10, 20, 30, 40)
fun(200, 300, 100)
fun(1, 2)
```

**206. How to return two numbers from a function?**

**Ans.**

```
def fun():
    return 10, 20
a, b = fun()
print("a={}, b={}".format(a, b))
```

The following will be the output of the preceding code:

a=10, b=20

**207. Create two list and iterate both lists simultaneously such that list1 should display item in the original order and list2 in the reverse order.**

**Ans.**

```
list1 = [10, 20, 30, 40]
list2 = [100, 200, 300, 400]
for x, y in zip(list1, list2[::-1]):
    print(x, y)
```

The following will be the output of the preceding code:

```
10 400
20 300
30 200
40 100
```

**Figure 54**

**208.****Write a program to remove empty elements from the list. Ans.**

```
b=["a","b","c","d"]
c=list(filter(None,b))
print(c)
```

The following will be the output of the preceding code:

```
['a', 'b', 'c', 'd']
```

*Figure 55***209. Write a program to input two string and create a new string by appending s2 in the middle of s1.****Ans.**

```
s1="HELLO"
s2="student"
middleIndex = int(len(s1) /2)
print("Original Strings are {} and {}".format(s1,
s2))
middleThree = s1[:middleIndex:]+ s2
+s1[middleIndex:]
print("After appending new string in middle",
middleThree)
```

The following will be the output of the preceding code:

```
Original Strings are HELLO and student
After appending new string in middle HEstudentLLO
```

*Figure 56***210. Create a function to count all lower case, upper case, digits, and special symbols from a given string.****Ans.**

```
def count_digit_symbol_char(input_string):
charCount = 0
digitCount = 0
```

```
symbolCount = 0
for char in input_string:
    if char.islower() or char.isupper():
        charCount+=1
    elif char.isnumeric():
        digitCount+=1
    else:
        symbolCount+=1

    print("Chars = ", charCount, "Digits = ",
          digitCount, "Symbol = ", symbolCount)
```

```
input_string = "B@eP&o3Si*4ti1V#e"
print("total counts of chars, digits, and symbols
\n")
```

```
count_digit_symbol_char(input_string)
```

The following will be the output of the preceding code:

```
total counts of chars, digits, and symbols
Chars = 10 Digits = 3 Symbol = 4
```

*Figure 57*

**211. Write a program to count the occurrence of all characters in a given string.**

**Ans.**

```
str1 = "banana"
countDict = dict()
for char in str1:
    count = str1.count(char)
    countDict[char]=count
print(countDict)
```

The following will be the output of the preceding code:

---

```
{'b': 1, 'a': 3, 'n': 2}
```

---

*Figure 58*

**212. Write a program to convert a string into a date-time object.**

**Ans.**

```
from datetime import datetime
```

```
date_string = "Jun 1 2020 10:20AM"
datetime_object = datetime.strptime(date_string, '%b
%d %Y %I:%M%p')
print(datetime_object)
```

The following will be the output of the preceding code:

---

```
2020-06-01 10:20:00
```

---

*Figure 59*

**213. Print the day of the week in a given date.**

**Ans.**

```
from datetime import datetime
```

```
given_date = datetime(2020, 6, 1)
```

```
print(given_date.strftime('%A'))
```

The following will be the output of the preceding code:

Monday

**214. Display today's date in a different format.**

**Ans.**

```
from datetime import date
```

```
today = date.today()
```

```
print("Today's date:", today)
```

```
# dd/mm/YY  
d1 = today.strftime("%d/%m/%Y")  
print("d1 =", d1)  
  
# Textual month, day and year  
d2 = today.strftime("%B %d, %Y")  
print("d2 =", d2)  
  
# mm/dd/y  
d3 = today.strftime("%m/%d/%y")  
print("d3 =", d3)  
  
# Month abbreviation, day and year  
d4 = today.strftime("%b-%d-%Y")  
print("d4 =", d4)  
  
print("Today's date and time",datetime.now())
```

The following will be the output of the preceding code:

```
Today's date: 2020-09-25  
d1 = 25/09/2020  
d2 = September 25, 2020  
d3 = 09/25/20  
d4 = Sep-25-2020  
Today's date and time 2020-09-25 16:39:25.482464
```

---

*Figure 60*

## **215. Print the current time in Python.**

**Ans.**

```
from datetime import datetime  
  
now = datetime.now().time() # time object  
  
print("now =", now)
```

## **216. Calculate the difference in years between two dates.**

**Ans. The first method**

```
from datetime import datetime  
start_date = datetime(2010,1,1)  
end_date = datetime(2012,1,1)  
difference = end_date - start_date  
difference_in_years = (difference.days + difference.  
seconds/86400)/365.2425  
print(round(difference_in_years))
```

### The second method

```
from datetime import datetime  
from dateutil.relativedelta import relativedelta  
datetime1 = datetime(2000, 1, 1)  
datetime2 = datetime(2010, 1, 1)  
time_difference = relativedelta(datetime2,  
datetime1)  
difference_in_years = time_difference.years  
print(difference_in_years, "years")
```

**217. Generate captcha. On every refresh, it should change. It must satisfy the following condition:**

**It must be eight characters long and contain two uppercase letters, one symbol, and a digit.**

**Ans.**

```
import  
random  
import string  
def captcha():  
    randomSource = string.ascii_letters + string.  
    digits + string.punctuation  
    capt = random.sample(randomSource, 4)  
    capt += random.sample(string.ascii_uppercase, 2)  
    capt += random.choice(string.digits)  
    capt+= random.choice(string.punctuation)  
  
    captcha_list = list(capt)
```

```
random.SystemRandom().shuffle(captcha_list)
capt = ''.join(captcha_list)
return capt

print ("Random captcha ",captcha())
```

**218. How to create a class without any variable and method and a method without the body?****Ans.**

```
class student:
    pass

def fun():
    pass
```

**219. How to check which class an object belongs to and check its base class?****Ans.**

```
class Vehicle:
    def __init__(self, name):
        self.name = name

class Bus(Vehicle):
    pass

class Car(Vehicle):
    pass
```

```
School_bus = Bus("Volvo")
cab=Car("Bugatti")
```

```
# use Python's built-in type() function
print(type(School_bus))
print(isinstance(School_bus, Vehicle))
print(type(cab))
print(isinstance(cab, Vehicle))
```

The following will be the output of the preceding code:

```
<class '__main__.Bus'>
True
<class '__main__.Car'>
True
```

**Figure 61**

**220. Remove items from set1 that are not common to both set1 and set2.**

**Ans.**

```
set1 = {10, 20, 30, 40, 50}
set2 = {30, 40, 50, 60, 70}

set1.intersection_update(set2)
print(set1)
```

**221. Get the key corresponding to the minimum value from the following dictionary:**

```
fruits={'apple':100,'kiwi':200,'banana':80,'pineapple':150}
```

**Ans.**

```
print(min(fruits,key=fruits.get))
```

**222. Create a 4X2 integer array and print its attribute.**

**Ans.**

```
import numpy

firstArray = numpy.empty([4,2], dtype = numpy.uint16)
print("Printing Array")
print(firstArray)

print("Printing numpy array Attributes")
print("1-- Array Shape is: ", firstArray.shape)
print("2-- Array dimensions are ", firstArray.ndim)
print("3-- Length of each element of array in bytes is ", firstArray.itemsize)
```

**223. Create a numpy array and print the value of the third column of all rows.**

**Ans.**

```
import numpy

sampleArray = numpy.array([[1,2,3], [11,22,33], [111,222,333],[1111,2222,3333]])
print("Printing Input Array")
print(sampleArray)

print("\n Printing array of items in the third
column from all rows")
newArray = sampleArray[:,2]
print(newArray)
```

**224. Create an 8X3 integer array from a range between 1 and 24 such that the difference between each element is 1, and then split the array into four equal-sized sub-arrays.**

**Ans.**

```
import numpy

print("Creating 8X3 array using numpy.arange")
sampleArray = numpy.arange(1, 25, 1)
sampleArray = sampleArray.reshape(8,3)
print (sampleArray)

print("\nDividing 8X3 array into 4 sub array\n")
subArrays = numpy.split(sampleArray, 4)
print(subArrays)
```

The following will be the output of the preceding code:

Creating 8X3 array using numpy.arange

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]
 [16 17 18]
 [19 20 21]
 [22 23 24]]
```

Dividing 8X3 array into 4 sub array

```
[array([[1, 2, 3],
       [4, 5, 6]]), array([[ 7,  8,  9],
       [10, 11, 12]]), array([[13, 14, 15],
       [16, 17, 18]]), array([[19, 20, 21],
       [22, 23, 24]])]
```

*Figure 62*

## 225. Find each company's highest price car.

| company       | company       | price   |
|---------------|---------------|---------|
| alfa-romero   | alfa-romero   | 16500.0 |
| audi          | audi          | 18920.0 |
| bmw           | bmw           | 41315.0 |
| chevrolet     | chevrolet     | 6575.0  |
| dodge         | dodge         | 6377.0  |
| honda         | honda         | 12945.0 |
| isuzu         | isuzu         | 6785.0  |
| jaguar        | jaguar        | 36000.0 |
| mazda         | mazda         | 18344.0 |
| mercedes-benz | mercedes-benz | 45400.0 |
| mitsubishi    | mitsubishi    | 8189.0  |
| nissan        | nissan        | 13499.0 |
| porsche       | porsche       | 37028.0 |
| toyota        | toyota        | 15750.0 |
| volkswagen    | volkswagen    | 9995.0  |
| volvo         | volvo         | 13415.0 |

*Figure 63*

**Ans.**

```
import pandas as pd  
df = pd.read_csv("Desktop\\Automobile_data.csv")  
car_Manufacturers = df.groupby('company')  
mileageDf = car_Manufacturers['company','average-  
mileage'].mean()  
print(mileageDf)
```

**226. Write a program to find the sum of series 5 +55+555+5555 ....N.**

**Ans.**

```
number_of_terms = 5  
start = 5  
sum = 0  
for i in range(0, number_of_terms):  
    print(start, end=" ")  
    sum += start  
    start = (start * 10) + 5  
print("\nSum of above series is:", sum)
```

The following will be the output of the preceding code:

```
5 55 555 5555 55555  
Sum of above series is: 61725
```

*Figure 64*

**227. Write a program to return the sum and average of the digits that appear in the string, ignoring all other characters.**

**Ans.**

```
import re  
  
inputStr = "English = 98 Science = 78 Math = 88  
History = 75"  
markList = [int(num) for num in re.findall(r'\b\d+\b', inputStr)]
```

```
totalMarks = 0  
for mark in markList:  
    totalMarks+=mark
```

```
percentage = totalMarks/len(markList)  
print("Total Marks is:", totalMarks, "Percentage is  
", percentage)
```

The following will be the output of the preceding code:

---

```
Total Marks is: 294 Percentage is 73.5
```

*Figure 65*





# TECHVIDYA

ISO 9001:2015 Accredited Company

*"Stay Updated, Stay Ahead"*

For TechVidya Candidates Only.  
Not For Selling Purpose.

