Python Interview Question & Answers

1. **What is Python?**
   List some popular applications of Python in the world of technology.
   Python is a widely-used general-purpose, high-level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.
   It is used for:
   - System Scripting
   - Web Development
   - Game Development
   - Software Development
   - Complex Mathematics

2. **What are the benefits of using Python language as a tool in the present scenario?**
   The following are the benefits of using Python language
   - Object-Oriented Language
   - High-Level Language
   - Dynamically Typed language
   - Extensive support Libraries
   - Presence of third-party modules
   - Open source and community development
   - Portable and Interactive
   - Portable across Operating systems

3. **Is Python a compiled language or an interpreted language?**
   Actually, Python is a partially compiled language and partially interpreted language. The compilation part is done first when we execute our code and this will generate byte code internally. This byte code gets converted by the Python virtual machine(p.v.m) according to the underlying platform(machine+operating system).

4. **What does the '#' Symbol in Python?**
   '#' is used to comment on everything that comes after on the line.

5. **What is the difference between a Mutable data type and an Immutable data type?**
   - Mutable data types can be edited i.e., they can change at runtime. Eg – List, Dictionary, etc.
   - Immutable data types can not be edited i.e., they can not change at runtime. Eg – String, Tuple, etc.

6. **How are arguments passed by value or by reference in Python?**
   Everything in Python is an object and all variables hold references to the objects. The reference values are according to the functions; as a result, you cannot change the value of the references. However, you can change the objects if they are mutable.

7. **What is the difference between a Set and Dictionary?**
    - The set is an unordered collection of data types that is iterable, mutable and has no duplicate elements.
    - A dictionary in Python is an unordered collection of data values, used to store data values like a map.

8. **What is List Comprehension? Give an Example.**
   List comprehension is a syntax construction to ease the creation of a list based on an existing iterable.
   For Example:

```
my_list = [i for i in range(1, 10)]
```

9. **What is a lambda function?**
   A lambda function is an anonymous function. This function can have any number of parameters but can have just one statement. For Example:

```
a = lambda x, y : x*y
print(a(7, 19))
```

   Output: 133

10. **What is a pass in Python?**
    Pass means performing no operation or in other words, it is a placeholder in the compound statement, where there should be a blank left and nothing has to be written there.

11. **What is the difference between / and // in Python?**
    // represents floor division whereas / represents precise division.

```
5//2 = 2
5/2 = 2.5
```

12. **How is Exceptional handling done in Python?**
    - There are 3 main keywords i.e. try, except, and finally which are used to catch exceptions and handle the recovering mechanism accordingly. Try is the block of a code that is monitored for errors. Except the block gets executed when an error occurs.
    - The beauty of the final block is to execute the code after trying for an error. This block gets executed irrespective of whether an error occurred or not. Finally block is used to do the required cleanup activities of objects/variables.

13. **What is a swapcase function in Python?**
    It is a string's function that converts all uppercase characters into lowercase and vice versa. It is used to alter the existing case of the string. This method creates a copy of the string which contains all the characters in the swap case.

```
string = "Hello Pune"
string.swapcase() ---> "hELLO pUNE"
```

14. Difference between for loop and while loop in Python

- The “for” Loop is generally used to iterate through the elements of various collection types such as List, Tuple, Set, and Dictionary. Developers use a “for” loop where they have both the conditions start and the end.
- The “while” loop is the actual looping feature that is used in any other programming language. Programmers use a Python while loop where they just have the end conditions.

15. Can we Pass a function as an argument in Python?

Yes, Several arguments can be passed to a function, including objects, variables (of the same or distinct data types), and functions. Functions can be passed as parameters to other functions because they are objects. Higher-order functions are functions that can take other functions as arguments.

16. What are *args and *kwargs?

To pass a variable number of arguments to a function in Python, use the special syntax *args and **kwargs in the function specification. It is used to pass a variable-length, keyword-free argument list. By using the *, the variable we associate with the * becomes iterable, allowing you to do operations on it such as iterating over it and using higher-order operations like map and filter.

17. Is Indentation Required in Python?

Yes, indentation is required in Python. A Python interpreter can be informed that a group of statements belongs to a specific block of code by using Python indentation. Indentations make the code easy to read for developers in all programming languages but in Python, it is very important to indent the code in a specific order.

18. What is Scope in Python?

The location where we can find a variable and also access it if required is called the scope of a variable.

- **Local variable:** Local variables are those that are initialized within a function and are unique to that function. It cannot be accessed outside of the function.
- **Global variables:** Global variables are the ones that are defined and declared outside any function and are not specified to any function.
- **Module-level scope:** It refers to the global objects of the current module accessible in the program.
- **Outermost scope:** It refers to any built-in names that the program can call. The name referenced is located last among the objects in this scope.

19. What is docstring in Python?

- Python documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods.
- Declaring Docstrings: The docstrings are declared using ”’triple single quotes”’ or “”“triple double quotes””” just below the class, method, or function declaration. All functions should have a docstring.
- Accessing Docstrings: The docstrings can be accessed using the __doc__ method of the object or using the help function.

20. What is a dynamically typed language?

Typed languages are the languages in which we define the type of data type and it will be known by the machine at the compile-time or at runtime. Typed languages can be classified into two categories:

- **Statically typed languages:** In this type of language, the data type of a variable is known at the compile time which means the programmer has to specify the data type of a variable at the time of its declaration.
- **Dynamically typed languages:** These are the languages that do not require any pre-defined data type for any variable as it is interpreted at runtime by the machine itself. In these languages, interpreters assign the data type to a variable at runtime depending on its value.

21. What is a break, continue, and pass in Python?

- The **break** statement is used to terminate the loop or statement in which it is present. After that, the control will pass to the statements that are present after the break statement, if available.
- **Continue** is also a loop control statement just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces the execution of the next iteration of the loop.
- **Pass** means performing no operation or in other words, it is a placeholder in the compound statement, where there should be a blank left and nothing has to be written there.

22. What are Built-in data types in Python?

The following are the standard or built-in data types in Python:

- **Numeric**: The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, a Boolean, or even a complex number.
- **Sequence Type**: The sequence Data Type in Python is the ordered collection of similar or different data types. There are several sequence types in Python:
- **String**
- **List**
- **Tuple**
- **Dictionary**
- **Range**
- **Mapping Types**: In Python, hashable data can be mapped to random objects using a mapping object. There is currently only one common mapping type, the dictionary, and mapping objects are mutable.
- **Set**: In Python, a Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

23. How do you floor a number in Python?

The Python math module includes a method that can be used to calculate the floor of a number.
floor() method in Python returns the floor of x i.e., the largest integer not greater than x.
ceil(x) in Python returns a ceiling value of x i.e., the smallest integer greater than or equal to x.

24. What is the difference between xrange and range functions?

range() and xrange() are two functions that could be used to iterate a certain number of times in for loops in Python. In Python 3, there is no xrange, but the range function behaves like xrange in Python 2.

- **range()** – This returns a list of numbers created using the range() function.
- **xrange()** – This function returns the generator object that can be used to display numbers only by looping. The only particular range is displayed on demand and hence called lazy evaluation.

25. What is Dictionary Comprehension? Give an Example

Dictionary Comprehension is a syntax construction to ease the creation of a dictionary based on the existing iterable.

```
my_dict = {i:1+7 for i in range(1, 10)}
```

26. Is Tuple Comprehension? If yes, how, and if not why?

```
(i for i in (1, 2, 3))
```

Tuple comprehension is not possible in Python because it will end up in a generator, not a tuple comprehension.

27. Differentiate between List and Tuple?

Let's analyze the differences between List and Tuple:

List

- Lists are Mutable data types.
- Lists consume more memory
- The list is better for performing operations, such as insertion and deletion.
- The implication of iterations is Time-consuming

Tuple

- Tuples are Immutable data types.
- Tuple consumes less memory as compared to the list
- A Tuple data type is appropriate for accessing the elements
- The implication of iterations is comparatively Faster

28. What is the difference between a shallow copy and a deep copy?

Shallow copy is used when a new instance type gets created and it keeps values that are copied whereas deep copy stores values that are already copied.

A shallow copy has faster program execution whereas a deep copy makes it slow.

29. Which sorting technique is used by sort() and sorted() functions of python?

Python uses the Tim Sort algorithm for sorting. It's a stable sorting whose worst case is **O(N log N)**. It's a hybrid sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data.

30. What are Decorators?

Decorators in simple terms is the specific change that we make in Python syntax to alter functions easily.

31. How do you debug a Python program?

- In Python, we can use the debugger pdb for debugging the code. To start debugging we have to enter the following lines on the top of a Python script.
  import pdb
  pdb.set_trace()
- After adding these lines, our code runs in debug mode. Now we can use commands like breakpoint, step through, step into etc for debugging.
- By using this command we can debug a Python program:
  $ python -m pdb python-script.py

32. What are Iterators in Python?

In Python, iterators are used to iterate a group of elements, containers like a list. Iterators are collections of items, and they can be a list, tuples, or a dictionary. Python iterator implements __itr__ and the next() method to iterate the stored elements. We generally use loops to iterate over the collections (list, tuple) in Python.

33. What are Generators in Python?

- In Python, the generator is a way that specifies how to implement iterators. It is a normal function except that it yields expression in the function. It does not implement __itr__ and next() method and reduces other overheads as well.
- If a function contains at least a yield statement, it becomes a generator. The yield keyword pauses the current execution by saving its states and then resumes from the same when required.

34. Does Python support multiple Inheritance?

Python does support multiple inheritances, unlike Java. Multiple inheritances mean that a class can be derived from more than one parent class.

35. What is Polymorphism and encapsulation in Python?

- Polymorphism means the ability to take multiple forms. So, for instance, if the parent class has a method named ABC then the child class also can have a method with the same name ABC having its own parameters and variables. Python allows polymorphism.
- Encapsulation means binding the code and the data together. A Python class is an example of encapsulation.

36. What is garbage collection in Python?

A garbage collection in Python manages the memory automatically and heap allocation. In simpler terms, the process of automatic deletion of unwanted or unused objects to free the memory is called garbage collection in Python.

37. How do you do data abstraction in Python?

Data Abstraction is providing only the required details and hides the implementation from the world. It can be achieved in Python by using interfaces and abstract classes.

38. How is memory management done in Python?

Python uses its private heap space to manage the memory. Basically, all the objects and data structures are stored in the private heap space. Even the programmer can not access this private space as the interpreter takes care of this space. 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.

39. How to delete a file using Python?

We can delete a file using Python by following approaches:

- os.remove()
- os.unlink()

40. What is slicing in Python?

Python Slicing is a string operation for extracting a part of the string, or some part of a list. With this operator, one can specify where to start the slicing, where to end, and specify the step. List slicing returns a new list from the existing list.

```
Lst[ Initial : End : IndexJump ]
```

41. What is a namespace in Python?

A namespace is a naming system used to make sure that names are unique to avoid naming conflicts.

42. What is PIP?

PIP is an acronym for **Python Installer Package** which provides a seamless interface to install various Python modules. It is a command-line tool that can search for packages over the internet and install them without any user interaction.

43. **What is a zip function?**

Python zip() function returns a zip object, which maps a similar index of multiple containers. It takes an iterable, converts it into an iterator and aggregates the elements based on iterables passed. It returns an iterator of tuples.

44. What are Pickling and Unpickling?

The Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using the dump function, this process is called pickling.
While the process of retrieving original Python objects from the stored string representation is called unpickling.
Python has a module named pickle. This module has the implementation of a powerful algorithm for serialization and deserialization of Python object structure.

45. How can we do Functional programming in Python?

In Functional Programming, we decompose a program into functions. These functions take input and after processing give an output. The function does not maintain any state.
Python provides built-in functions that can be used for Functional

programming. Some of these functions are:

- **Map()**
- **reduce()**
- **filter()**

Event iterators and generators can be used for Functional programming in Python.

46. What is __init__() in Python?

Equivalent to constructors in OOP terminology, __init__ is a reserved method in Python classes. The __init__ method is called automatically whenever a new object is initiated. This method allocates memory to the new object as soon as it is created. This method can also be used to initialize variables.

47. Write a code to display the current time?

```
current_time= time.localtime()
print ("Current time is", current_time)
```

48. What are Access Specifiers in Python?

Python uses the '_' symbol to determine the access control for a specific data member or a member function of a class. A Class in Python has three types of Python access modifiers:

- **Public Access Modifier**: The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.
- **Protected Access Modifier:** The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.
- **Private Access Modifier**: The members of a class that are declared private are accessible within the class only, the private access modifier is the most secure access modifier. Data members of a class are declared private by adding a double underscore '__' symbol before the data member of that class.

49. What are unit tests in Python?

Unit Testing is the first level of software testing where the smallest testable parts of the software are tested. This is used to validate that each unit of the software performs as designed. The unit test framework is Python's xUnit style framework. The White Box Testing method is used for Unit testing.

50. Python Global Interpreter Lock (GIL)?

Python Global Interpreter Lock (GIL) is a type of process lock that is used by Python whenever it deals with processes. Generally, Python only uses only one thread to execute the set of written statements. The performance of the single-threaded process and the multi-threaded process will be the same in Python and this is because of GIL in Python. We can not achieve multithreading in Python because we have a global interpreter lock that restricts the threads and works as a single thread.

51. What are Function Annotations in Python?

- Function Annotation is a feature that allows you to add metadata to function parameters and return values. This way you can specify the input type of the function parameters and the return type of the value the function returns.
- Function annotations are arbitrary Python expressions that are associated with various parts of functions. These expressions are evaluated at compile time and have no life in Python's runtime environment. Python does not attach any meaning to these annotations. They take life when interpreted by third-party libraries, for example, mypy.

52. What are Exception Groups in Python?

- The ExceptionGroup can be handled using a new except* syntax. The * symbol indicates that multiple exceptions can be handled by each except* clause.
- ExceptionGroup is a collection/group of different kinds of Exception. Without creating Multiple Exceptions we can group together different Exceptions which we can later fetch one by one whenever necessary, the order in which the Exceptions are stored in the Exception Group doesn't matter while calling them.

```
try:
    raise ExceptionGroup('Example ExceptionGroup', (
        TypeError('Example TypeError'),
        ValueError('Example ValueError'),
        KeyError('Example KeyError'),
        AttributeError('Example AttributeError')
    ))
except* TypeError:
    ...
except* ValueError as e:
    ...
except* (KeyError, AttributeError) as e:
    ...
```

53. What is Python Switch Statement

Python has implemented a switch case feature called "structural pattern matching". You can implement this feature with the match and case keywords. Note that the underscore symbol is what you use to define a default case for the switch statement in Python.

Note: Before Python 3.10 Python didn't support match Statements.

```
match term:
    case pattern-1:
        action-1
    case pattern-2:
        action-2
    case pattern-3:
        action-3
    case _:
```

```
action-default
```

54. What is the improvement in the enumerate() function of Python?

In Python, enumerate() function is an improvement over regular iteration.
The enumerate() function returns an iterator that gives (0, item[0]).

```
>>> thelist=['a','b']
>>> for i,j in enumerate(thelist):
... print i,j
...
0 a
1 b
```

55. uses and benefits of python

- High readability
- reduces cost of program maintenance
- open source
- support third party packages modularity
- code reverse.

56. Dynamically typed language?

Typing refers to type checking in programming language
storing type language such as a Python 1 + 2 will result in type error since this language don't allow for “type coercion” (implicit conversion of data types) on the other hand
a weekly type data weakly typed language such as JS will simply output 12 to a result
Two stage of a Typing-checking

- **Static-** data types are checked before execution.
- **dynamic-** data type are checked during execution

57. What is self in code?

self represents the instances of class . These handy keywords allow you to access the variables, attributes and methods of a defined class in Python.
Self parameter doesn't have to be you named ‘self’ as you can call it by any other home however the self parameter must always be the first parameter of any class function
regardless of a name choose chosen so instead of selfie you call you ‘mine’ or ‘ours’ or anything else

```
Class address:
        def __init__(mine,street,number):
                Mine.street = street
                Mine.number = number
        Def myfunc(abc):
                print(“my address is “ abc,street)

P1 = address(“Albert street”,20)
p1.myfunc()
```

58. What is comprehension in Python?

It provide us with the short and a concise way to construct a new sequences
[such as a list set dictionary etc]
using sequences which have been already defined Python supports
four types of a comprehension

- list comprehension
- dictionary comprehension
- set comprehension
- generator comprehension

A. **List comprehension-**

- it provide an elegant way to create a new list.The following is the basic structure of a list comprehension
- output list= [output_execution for var in input_list if (var satisfy this condition)]
- Note - list comprehension may or may not contain and if condition list comprehension can contain multiple for nested list conference comprehension

1. example one suppose we want to create an output list which contains only the even number which are present in the input list let's see how to do this using for loops and list comprehension and decide which method suits better

```
#without using list comprehension
Input_list = [1, 2, 3, 4, 4, 5, 6, 7,7]
Output_list=[ ]
#look for constructing output
 for var in input_list:
    If var % 2 ==0:
      Output_list.append(var)
print("Output list using for loop, output_list)
```

Output = output list using for loop : [2 4 4 6]

2 Suppose we want to create an output list which contains squares of all the numbers from 1 to 9 let's see how to do this using for loops and list comprehension

```
# Constructing output list using for loop
 output list is equal to[]
 for war in range(1, 10):
        output_list.append(var ** 2):
print("Output list using for loops: ",output_list )
```

output = output list using for loop : [1 4 9 16 25 36 49 64 81]

```
#using list comprehension
list_using_comprehension=[ var * 2 for var in range(1,10)]
print("Output list using for loops: ", list_using_comprehension )
```

output = output list using for comprehension: [1 4 9 16 25 36 49 64 81]

B. Dictionary comprehension

Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

```
output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}
```

Example #1: Suppose we want to create an output dictionary which contains only the odd numbers that are present in the input list as keys and their cubes as values. Let's see how to do this using for loops and dictionary comprehension.

```
input_list = [1, 2, 3, 4, 5, 6, 7]
output_dict = {}
 # Using loop for constructing output dictionary
for var in input_list:
    if var % 2 != 0:
        output_dict[var] = var**3
  print("Output Dictionary using for loop:", output_dict )
```

Output: Output Dictionary using for loop: {1: 1, 3: 27, 5: 125, 7: 343}

```
# Using Dictionary comprehensions
# for constructing output dictionary
  input_list = [1,2,3,4,5,6,7]
dict_using_comp = {var:var ** 3 for var in input_list if var % 2 != 0}
print("Output Dictionary using dictionary comprehensions:", dict_using_comp)
```

Output: Output Dictionary using dictionary comprehensions: {1: 1, 3: 27, 5: 125, 7: 343}

Example #2: Given two lists containing the names of states and their corresponding capitals, construct a dictionary which maps the states with their respective capitals. Let's see how to do this using for loops and dictionary comprehension.

```
state = ['Gujarat', 'Maharashtra', 'Rajasthan']
capital = ['Gandhinagar', 'Mumbai', 'Jaipur']
output_dict = {}
# Using loop for constructing output dictionary
for (key, value) in zip(state, capital):
output_dict[key] = value
print("Output Dictionary using for loop:", output_dict)
```

Output Output Dictionary using for loop: {'Gujarat': 'Gandhinagar', 'Maharashtra': 'Mumbai', 'Rajasthan': 'Jaipur'}

```
# Using Dictionary comprehensions
# for constructing output dictionary
state = ['Gujarat', 'Maharashtra', 'Rajasthan']
capital = ['Gandhinagar', 'Mumbai', 'Jaipur']
dict_using_comp = {key:value for (key, value) in zip(state, capital)}
```

```
print("Output Dictionary using dictionary comprehensions:", dict_using_comp)
```

Output: Output Dictionary using dictionary comprehensions: {'Rajasthan': 'Jaipur', 'Maharashtra': 'Mumbai', 'Gujarat': 'Gandhinagar'}

C. set comprehension:

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { }. Let's look at the following example to understand set comprehensions.

Example #1 : Suppose we want to create an output set which contains only the even numbers that are present in the input list. Note that set will discard all the duplicate values. Let's see how we can do this using for loops and set comprehension.

```
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
output_set = set()
# Using loop for constructing output set
for var in input_list:
    if var % 2 == 0:
        output_set.add(var)
print("Output Set using for loop:", output_set)
```

Output: Output Set using for loop: {2, 4, 6}

```
# Using Set comprehensions
# for constructing output set
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
set_using_comp = {var for var in input_list if var % 2 == 0}
print("Output Set using set comprehensions:",set_using_comp)
```

Output: Output Set using set comprehensions: {2, 4, 6}

D. generator comprehension

Generator Comprehensions are very similar to list comprehensions. One difference between them is that generator comprehensions use circular brackets whereas list comprehensions use square brackets. The major difference between them is that generators don't allocate memory for the whole list. Instead, they generate each value one by one which is why they are memory efficient. Let's look at the following example to understand generator comprehension:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_gen = (var for var in input_list if var % 2 == 0)
print("Output values using generator comprehensions:", end = ' ')
for var in output_gen:
    print(var, end = ' ')
```

Output: Output values using generator comprehensions: 2 4 4 6

59. What is a decorator?

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate.

- **Assigning Functions to Variables**

To kick us off we create a function that will add one to a number whenever it is called. We'll then assign the function to a variable and use this variable to call the function.

```
def plus_one(number):
    return number + 1
add_one = plus_one
add_one(5)
```

Output:- 6

- **Defining Functions Inside other Functions**

Next, we'll illustrate how you can define a function inside another function in Python. Stay with me, we'll soon find out how all this is relevant in creating and understanding decorators in Python.

```
def plus_one(number):
    def add_one(number):
        return number + 1
    result = add_one(number)
    return result
plus_one(4)
```

Output 5

- **Passing Functions as Arguments to other Functions**

Functions can also be passed as parameters to other functions. Let's illustrate that below.

```
def plus_one(number):
    return number + 1
def function_call(function):
    number_to_add = 5
    return function(number_to_add)
function_call(plus_one)
```

Output 6

- **Functions Returning other Functions**

A function can also generate another function. We'll show that below using an example.

```
def hello_function():
    def say_hi():
        return "Hi"
    return say_hi
hello = hello_function()
hello()
```

Output 'Hi'

- **Nested Functions have access to the Enclosing Function's Variable Scope**

Python allows a nested function to access the outer scope of the enclosing function. This is a critical concept in decorators -- this pattern is known as a Closure.

```
def print_message(message):
    "Enclosing Function"
    def message_sender():
        "Nested Function"
        print(message)
    message_sender()
print_message("Some random message")
```

Output Some random message

- **Creating Decorators**

With these prerequisites out of the way, let's go ahead and create a simple decorator that will convert a sentence to uppercase. We do this by defining a wrapper inside an enclosed function. As you can see it is very similar to the function inside another function that we created earlier.

```
def uppercase_decorator(function):
    def wrapper():
        func = function()
        make_uppercase = func.upper()
        return make_uppercase
    return wrapper
```

Our decorator function takes a function as an argument, and we shall, therefore, define a function and pass it to our decorator. We learned earlier that we could assign a function to a variable. We'll use that trick to call our decorator function.

```
def say_hi():
    return 'hello there'
decorate = uppercase_decorator(say_hi)
decorate()
```

Output 'HELLO THERE'

However, Python provides a much easier way for us to apply decorators. We simply use the @ symbol before the function we'd like to decorate. Let's show that in practice below.

```
@uppercase_decorator
def say_hi():
    return 'hello there'
say_hi()
```

Output 'HELLO THERE'

- **Applying Multiple Decorators to a Single Function**

We can use multiple decorators to a single function. However, the decorators will be applied in the order that we've called them. Below we'll define another decorator that splits the sentence into a list. We'll then apply the uppercase_decorator and split_string decorator to a single function.

```
def split_string(function):
    def wrapper():
        func = function()
        splitted_string = func.split()
        return splitted_string
    return wrapper

@split_string
@uppercase_decorator
def say_hi():
    return 'hello there'
say_hi()

['HELLO', 'THERE']
```

From the above output, we notice that the application of decorators is from the bottom up. Had we interchanged the order, we'd have seen an error since lists don't have an upper attribute. The sentence has first been converted to uppercase and then split into a list.

- **Accepting Arguments in Decorator Functions**

Sometimes we might need to define a decorator that accepts arguments. We achieve this by passing the arguments to the wrapper function. The arguments will then be passed to the function that is being decorated at call time.

```
def decorator_with_arguments(function):
    def wrapper_accepting_arguments(arg1, arg2):
        print("My arguments are: {0}, {1}".format(arg1,arg2))
        function(arg1, arg2)
    return wrapper_accepting_arguments

@decorator_with_arguments
def cities(city_one, city_two):
    print("Cities I love are {0} and {1}".format(city_one, city_two))

cities("Nairobi", "Accra")
```

Output My arguments are: Nairobi, Accra
Cities I love are Nairobi and Accra

- **Defining General Purpose Decorators**

To define a general purpose decorator that can be applied to any function we use args and **kwargs. args and **kwargs collect all positional and keyword arguments and store them in the args and kwargs variables. args and kwargs allow us to pass as many arguments as we would like during function calls.

```
def a_decorator_passing_arbitrary_arguments(function_to_decorate):
    def a_wrapper_accepting_arbitrary_arguments(*args,**kwargs):
        print('The positional arguments are', args)
        print('The keyword arguments are', kwargs)
        function_to_decorate(*args)
    return a_wrapper_accepting_arbitrary_arguments

@a_decorator_passing_arbitrary_arguments
def function_with_no_argument():
    print("No arguments here.")

function_with_no_argument()

The positional arguments are ()
The keyword arguments are {}
No arguments here.
```

Let's see how we'd use the decorator using positional arguments.

```
@a_decorator_passing_arbitrary_arguments
def function_with_arguments(a, b, c):
    print(a, b, c)
function_with_arguments(1,2,3)

The positional arguments are (1, 2, 3)
The keyword arguments are {}
1 2 3
```

Keyword arguments are passed using keywords. An illustration of this is shown below.

```
@a_decorator_passing_arbitrary_arguments
def function_with_keyword_arguments():
    print("This has shown keyword arguments")

function_with_keyword_arguments(first_name="Derrick", last_name="Mwiti")
```

```
The positional arguments are ()
The keyword arguments are {'first_name': 'Derrick', 'last_name': 'Mwiti'}
This has shown keyword arguments
```

- Passing Arguments to the Decorator

Now let's see how we'd pass arguments to the decorator itself. In order to achieve this, we define a decorator maker that accepts arguments then defines a decorator inside it. We then define a wrapper function inside the decorator as we did earlier.

```
def decorator_maker_with_arguments(decorator_arg1, decorator_arg2, decorator_arg3):
    def decorator(func):
        def wrapper(function_arg1, function_arg2, function_arg3) :
```

```
        "This is the wrapper function"
        print("The wrapper can access all the variables\n"
              "\t- from the decorator maker: {0} {1} {2}\n"
              "\t- from the function call: {3} {4} {5}\n"
              "and pass them to the decorated function"
              .format(decorator_arg1, decorator_arg2,decorator_arg3,
                      function_arg1, function_arg2,function_arg3))
        return func(function_arg1, function_arg2,function_arg3)
    return wrapper
  return decorator

pandas = "Pandas"
@decorator_maker_with_arguments(pandas, "Numpy","Scikit-learn")
def decorated_function_with_arguments(function_arg1, function_arg2,function_arg3):
    print("This is the decorated function and it only knows about its arguments: {0}"
          " {1}" " {2}".format(function_arg1, function_arg2,function_arg3))

decorated_function_with_arguments(pandas, "Science", "Tools")
```

```
The wrapper can access all the variables
    - from the decorator maker: Pandas Numpy Scikit-learn
    - from the function call: Pandas Science Tools
and pass them to the decorated function
This is the decorated function, and it only knows about its arguments: Pandas Science Tools
```

- **Debugging Decorators**

As we have noticed, decorators wrap functions. The original function name, its docstring, and parameter list are all hidden by the wrapper closure: For example, when we try to access the decorated_function_with_arguments metadata, we'll see the wrapper closure's metadata. This presents a challenge when debugging.

```
        decorated_function_with_arguments.__name__
        'wrapper'
        decorated_function_with_arguments.__doc__
        'This is the wrapper function'
```

In order to solve this challenge Python provides a functools.wraps decorator. This decorator copies the lost metadata from the undecorated function to the decorated closure. Let's show how we'd do that.

```
        import functools

        def uppercase_decorator(func):
            @functools.wraps(func)
            def wrapper():
                return func().upper()
            return wrapper
        @uppercase_decorator
```

```
def say_hi():
    "This will say hi"
    return 'hello there'

say_hi()
'HELLO THERE'
```

When we check the say_hi metadata, we notice that it is now referring to the function's metadata and not the wrapper's metadata.

say_hi.__name__
'say_hi'
say_hi.__doc__
'This will say hi'

It is advisable and good practice to always use functools.wraps when defining decorators. It will save you a lot of headache in debugging.

- **Python Decorators Summary**

Decorators dynamically alter the functionality of a function, method, or class without having to directly use subclasses or change the source code of the function being decorated. Using decorators in Python also ensures that your code is DRY(Don't Repeat Yourself). Decorators have several use cases such as:

Authorization in Python frameworks such as Flask and Django
Logging
Measuring execution time
Synchronization

60. What are Python Modules?

Files containing Python codes are referred to as Python Modules. This code can either be classes, functions, or variables and saves the programmer time by providing the predefined functionalities when needed. It is a file with “.py” extension containing an executable code. Commonly used built modules are listed below:
os
sys
data time
math
random
JSON

61. What is pep 8?

PEP in Python stands for Python Enhancement Proposal. It is a set of rules that specify how to write and design Python code for maximum readability.

62. What are Python packages?

A Python package refers to the collection of different sub-packages and modules based on the similarities of the function.

63. What are the popular Python libraries used in Data analysis?

Some of the popular libraries of Python used for Data analysis are:

- **Pandas**: Powerful Python Data Analysis Toolkit
- **SciKit**: This is a machine learning library in Python.
- **Seaborn**: This is a statistical data visualization library in Python.
- **SciPy**: This is an open source system for science, mathematics and
- engineering implemented in Python.

64. What is the difference between append() and extend() methods?

Both append() and extend() methods are methods used to add elements at the end of a list.
append(element): Adds the given element at the end of the list that called this append() method
extend(another-list): Adds the elements of another list at the end of the list that called this extend() method

65. What is docstring in Python?

Python lets users include a description (or quick notes) for their methods using documentation strings or docstrings. Docstrings are different from regular comments in Python as, rather than being completely ignored by the Python Interpreter like in the case of comments, these are defined within triple quotes.

```
"""
Using docstring as a comment.
This code add two numbers
"""
x=7
y=9
z=x+y
print(z)
```

66. How is Multithreading achieved in Python?

Python has a multi-threading package ,but commonly not considered as good practice to use it as it will result in increased code execution time.
Python has a constructor called the Global Interpreter Lock (GIL). The GIL ensures that only one of your ‘threads’ can execute at one time.The process makes sure that a thread acquires the GIL, does a little work, then passes the GIL onto the next thread.
This happens at a very Quick instance of time and that’s why to the human eye it seems like your threads are executing parallely, but in reality they are executing one by one by just taking turns using the same CPU core.

67. What are the common built-in data types in Python?

Python supports the below-mentioned built-in data types:

- Immutable data types:

Number , String, Tuple

- Mutable data types:

List, Dictionary, Set

68. What is the difference between split() and slicing in Python?

Both split() function and slicing work on a String object. By using split() function, we can get the list of words from a String.

E.g. 'a b c '.split() returns [‘a’, ‘b’, ‘c’]

Slicing is a way of getting substring from a String. It returns another String.

E.g. >>> 'a b c'[2:3] returns b

69. How can you randomize the items of a list in place in Python?

This can be easily achieved by using the Shuffle() function from the random library as shown below:

```
from random import shuffle
List = ['He', 'Loves', 'To', 'Code', 'In', 'Python']
shuffle(List)
print(List)
```

Output: [‘Loves’,’He’ ,’To ,’In’, ‘Python’,’Code’]

70. What are negative indexes and why are they used?

To access an element from ordered sequences, we simply use the index of the element, which is the position number of that particular element. The index usually starts from 0, i.e., the first element has index 0, the second has 1, and so on.

Python Indexing

When we use the index to access elements from the end of a list, it’s called reverse indexing. In reverse indexing, the indexing of elements starts from the last element with the index number ‘−1’. The second last element has index ‘−2’, and so on. These indexes used in reverse indexing are called negative indexes.

71. Explain split(), sub(), subn() methods of “re” module in Python?

These methods belong to the Python RegEx or ‘re’ module and are used to modify strings.

split(): This method is used to split a given string into a list.

sub(): This method is used to find a substring where a regex pattern matches, and then it replaces the matched substring with a different string.

subn(): This method is similar to the sub() method, but it returns the new string, along with the number of replacements.

72. What is a map function in Python?

The map() function in Python has two parameters, function and iterable. The map() function takes a function as an argument and then applies that function to all the elements of an iterable, passed to it as another argument. It returns an object list of results.

For example:

```
def calculateSq(n):
return n*n
```

```
numbers = (2, 3, 4, 5)
result = map( calculateSq, numbers)
print(result)
```

73. Explain all file processing modes supported in Python?

Python has various file processing modes.

For opening files, there are three modes:

- read-only mode (r)
- write-only mode (w)
- read–write mode (rw)

For opening a text file using the above modes, we will have to append ‘t’ with them as follows:

- read-only mode (rt)
- write-only mode (wt)
- read–write mode (rwt)

Similarly, a binary file can be opened by appending ‘b’ with them as follows:

- read-only mode (rb)
- write-only mode (wb)
- read–write mode (rwb)

To append the content in the files, we can use the append mode (a):

For text files, the mode would be ‘at’

For binary files, it would be ‘ab’

74. How will you remove duplicate elements from a list?

To remove duplicate elements from the list we use the set() function.

Consider the below example:

```
demo_list=[5,4,4,6,8,12,12,1,5]
 unique_list = list(set(demo_list))
output:[1,5,6,8,12]
```

75. How will you read a random line in a file?

We can read a random line in a file using the random module.

For example:

```
import random
def read_random(fname):
lines = open(fname).read().splitlines()
return random.choice(lines)
print(read_random (‘hello.txt’))
```

76. How can files be deleted in Python?

You need to import the OS Module and use os.remove() function for deleting a file in python.

consider the code below:

```
import os
os.remove("file_name.txt")
```

77. How can you generate random numbers in Python?

This is achieved with importing the random module,it is the module that is used to generate random numbers.
Syntax:
import random
random.random # returns the floating point random number between the range of [0,1].

78. What is slicing in Python?

Slicing is a process used to select a range of elements from sequence data type like list, string and tuple. Slicing is beneficial and easy to extract out the elements. It requires a : (colon) which separates the start index and end index of the field. All the data sequence types List or tuple allows us to use slicing to get the needed elements. Although we can get elements by specifying an index, we get only a single element whereas using slicing we can get a group or appropriate range of needed elements.
Syntax:

```
List_name[start:stop]
```

79. Define Constructor in Python?

Constructor is a special type of method with a block of code to initialize the state of instance members of the class. A constructor is called only when the instance of the object is created. It is also used to verify that they are sufficient resources for objects to perform a specific task.
There are two types of constructors in Python, and they are:

- Parameterized constructor
- Non-parameterized constructor

80. How can we create a constructor in Python programming?

The _init_ method in Python stimulates the constructor of the class. Creating a constructor in Python can be explained clearly in the below example.

```
class Student:
def __init__(self,name,id):
self.id = id;
self.name = name;
def display (self):
print("ID: %d nName: %s"%(self.id,self.name))
stu1 =Student("nirvi",105)
stu2 = Student("tanvi",106)
#accessing display() method to print employee 1 information
stu1.display();
#accessing display() method to print employee 2 information
stu2.display();
```

Output:
ID: 1
Name: nirvi
ID: 106
Name: Tanvi