1. What is PEP 8 and why is it useful?

- PEP is an acronym for Python Enhancement Proposal.
- It is an official design document that contains a set of rules specifying how to format Python code and helps to achieve maximum readability.
- PEP 8 is useful because it documents all style guides for Python code.
- This is because contributing to the Python open-source community requires you to adhere to these style guidelines strictly

2. What is scope in Python?

- A scope in Python is a block of code in which a Python code lives.
- Global scope: It refers to the top-most scope in a program. These are global variables available throughout the code execution since their inception in the main body of the Python code.
- Local scope: It refers to the local variables available inside a current function and can only be used locally.

3. Is Python considered a programming or a scripting language?

- Python is generally considered to be a general-purpose programming language, but we can also use it for scripting.
- A scripting language is also a programming language that is used for automating a repeated task that involves similar types of steps while executing a program.
- Filename extensions for Python scripting language are of different types, such as `.py, .pyc, .pyd, .pyo, .pyw, and .pyz.`

4. What are keywords in Python?

- Keywords in Python are reserved words with a special meaning.
- They are generally used to define different types of variables.
- We cannot use keywords in place of variable or function names.

- There are 35 reserved keywords in Python

5. What is init ?

- In Python, init is a method or constructor used to automatically allocate memory when a new object or instance of a class is created.
- All classes have the `__init__` method included.

6. What are iterators in Python?

- An iterator in Python is an object used to iterate over a finite number of elements in data structures like lists, tuples, dicts, and sets.
- Iterators allow us to traverse through all the elements of a collection and return a single element at a time.
- The iterator object is initialized via the `iter()` method and uses the `next()` method for iteration.

7. What is a Python decorator?

- A decorator in Python is a function that allows a user to add a new piece of functionality to an existing object without modifying its structure.
- You usually call decorators before the definition of the function you want to decorate.

8. What are pickling and unpickling?

- **Pickling** is a process in Python where an object hierarchy is transformed into a byte stream. Pickling is also known as “serialization” or “marshaling”
- **Unpickling**, in contrast, is the opposite of pickling. It happens when a byte stream is converted back into an object hierarchy.
- Pickling uses the pickle module in Python. This module has the method `pickle.dump()` to dump Python objects to disks to achieve pickling.
- Unpickling uses the method `pickle.load()` to retrieve the data as Python objects.

9.What is the difference between .py and .pyc files?

- The .py files are the python source code files. While the .pyc files contain the bytecode of the python files.
- .pyc files are created when the code is imported from some other source.
- The interpreter converts the source .py files to .pyc files which helps by saving time.

10.What is slicing in Python?

- Slicing is used to access parts of sequences like lists, tuples, and strings.
- The syntax of slicing is-`[start:end:step]`. The step can be omitted as well.
- When we write `[start:end]` this returns all the elements of the sequence from the start (inclusive) till the end-1 element.

11. What is the use of the pass keyword in Python?

- Pass is a null statement that does nothing.
- It is often used as a placeholder where a statement is required syntactically, but no action needs to be taken.
- For example, if you want to define a function or a class but haven't yet decided what it should do, you can use the pass as a placeholder.

12. What is the use of the continue keyword in Python?

- Continue is used in a loop to skip over the current iteration and move on to the next one.
- When continue is encountered, the current iteration of the loop is terminated, and the next one begins.

13.What is the use of try and except block in Python?

- The try and except blocks in Python are used to handle exceptions. An exception is an error that occurs during the execution of a program.
- The try block contains code that might cause an exception to be raised. The except block contains code that is executed if an exception is raised during the execution of the try block.
- Using a try-except block will save the code from an error to occur and can be executed with a message or output we want in the except block.

14.What are Python functions, and how do they help in code optimization?

- In Python, a function is a block of code that can be called by other parts of your program. Functions are useful because they allow you to reuse code and divide your code into logical blocks that can be tested and maintained separately.
- To call a function in Python, you simply use the function name followed by a pair of parentheses and any necessary arguments. The function may or may not return a value that depends on the usage of the turn statement.

Functions can also help in code optimization:

- Code reuse: Functions allow you to reuse code by encapsulating it in a single place and calling it multiple times from different parts of your program. This can help to reduce redundancy and make your code more concise and easier to maintain.
- Improved readability: By dividing your code into logical blocks, functions can make your code more readable and easier to understand. This can make it easier to identify bugs and make changes to your code.
- Easier testing: Functions allow you to test individual blocks of code separately, which can make it easier to find and fix bugs.
- Improved performance: Functions can also help to improve the performance of your code by allowing you to use optimized code libraries or by allowing the Python interpreter to optimize the code more effectively.

15. What is the difference between return and yield keywords?

- Return is used to exit a function and return a value to the caller. When a return statement is encountered, the function terminates immediately, and the value of the expression following the return statement is returned to the caller.
- Yield, on the other hand, is used to define a ***generator*** function. A generator function is a special kind of function that produces a sequence of values one at a time instead of returning a single value. When a yield statement is encountered, the generator function produces a value and suspends its execution, saving its state for later.

16.What is the use of the ‘assert’ keyword in Python?

- In Python, the assert statement is used to test a condition. If the condition is True, then the program continues to execute. If the condition is False, then the program raises an AssertionError exception.
- The assert statement is often used to check the internal consistency of a program. For example, you might use an assert statement to check that a list is sorted before performing a binary search on the list.
- It’s important to note that the assert statement is used for debugging purposes and is not intended to be used as a way to handle runtime errors. In production code, you should use try and except blocks to handle exceptions that might be raised at runtime.

17.Where can we use a tuple instead of a list?

- We can use tuples as dictionary keys as they are hashable. Since tuples are immutable, it is safer to use if we don’t want values to change.
- Tuples are faster and have less memory, so we can use tuples to access only the elements.

18. What is inheritance, and how is it useful?

- With inheritance, we can use the attributes and methods of the parent class in the child class.
- We can also modify the parent class methods to suit the child class.

They are different types of inheritance supported by Python:

- Single Inheritance – where a derived class acquires the members of a single superclass.
- Multi-level inheritance – a derived class d1 is inherited from base class base1, and d2 is inherited from base2.
- Hierarchical inheritance – from one base class you can inherit any number of child classes
- Multiple inheritances – a derived class is inherited from more than one base class.

19.Is indentation required in python?

- 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, it will not execute accurately and will throw errors as well.

20.What is the difference between Python Arrays and lists?

- Arrays and lists, in Python, have the same way of storing data. But, arrays can hold only a single data type elements whereas lists can hold any data type elements.

21.What are Python libraries? Name a few of them.

Python libraries are a collection of Python packages. Some of the majorly used python libraries are – Numpy, Pandas, Matplotlib, Scikit-learn and many more.

22.What is Polymorphism 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.

23.Define encapsulation in Python?

Encapsulation means binding the code and the data together. A Python class in an example of encapsulation

24.What is Pythonpath?

- A Pythonpath tells the Python interpreter to locate the module files that can be imported into the program.
- It includes the Python source library directory and source code directory. You can preset Pythonpath as a Python installer

25.What is the maximum possible length of an identifier in Python?

- The maximum possible length of an identifier is the maximum length of 79 characters.

26.How is memory managed in Python?

- Memory management in Python is handled by the Python Memory Manager. The memory allocated by the manager is in form of a private heap space dedicated to Python. All Python objects are stored in this heap and being private, it is inaccessible to the programmer. Though, python does provide some core API functions to work upon the private heap space.
- Additionally, Python has an in-built garbage collection to recycle the unused memory for the private heap space.

27.What is the difference between .py and .pyc files?

- .py files contain the source code of a program. Whereas, .pyc file contains the bytecode of your program. We get bytecode after compilation of .py file (source code). .pyc files are not created for all the files that you run. It is only created for the files that you import.
- Before executing a python program python interpreter checks for the compiled files. If the file is present, the virtual machine executes it. If not found, it checks for .py file. If found, compiles it to .pyc file and then python virtual machine executes it.
- Having .pyc file saves you the compilation time

28. Write a Program to add two integers >0 without using the plus operator.

```
def add_nums(num1, num2):
  while num2 != 0:
      data = num1 & num2
      num1 = num1 ^ num2
      num2 = data << 1
  return num1
print(add_nums(2, 10))#12
```

29.Write a program to check and return the pairs of a given array A whose sum value is equal to a target value N.

```
def print_pairs(arr, N):
  # hash set
  hash_set = set()#we can use [ ] also

  for i in range(0, len(arr)):
      val = N-arr[i]
      if (val in hash_set):    #check if N-x is there in set, print the pair
          print("Pairs " + str(arr[i]) + ", " + str(val))
      hash_set.add(arr[i])

# driver code
arr = [1, 2, 40, 3, 9, 4]
N = 3
print_pairs(arr, N)
```

30.Write a program) which takes a sequence of numbers and check if all numbers are unique.

```
def check_distinct(data_list):
 if len(data_list) == len(set(data_list)):
   return True
 else:
   return False;
print(check_distinct([1,6,5,8]))     #Prints True
print(check_distinct([2,2,5,5,7,8])) #Prints False
```

31.What is the purpose of modules and packages in Python?

- Modules group related functions and variables.
- Packages group modules into hierarchies.

- Used for code organization and import for reuse.

32.What are init and str methods used for?

init: This is the constructor method called when an object is created. It's used to initialize the object's attributes with specific values or perform necessary setup.

str: This method defines how the object is represented when printed or converted to a string. It allows you to customize the object's representation for better readability or information display.

32.What is the purpose of generators in Python?

- Functions that create iterators, producing values one at a time instead of storing the entire sequence in memory.
- Efficient for iterating over large datasets or situations where generating all elements upfront is unnecessary.
- Use yield keyword to return each element sequentially

33. What is the difference between `staticmethod`, `classmethod`, and instance methods in Python?

- **`*staticmethod*`:** A static method is a method that belongs to a class rather than an instance of the class. It does not have access to instance or class variables and is defined using the `@staticmethod` decorator.
- **`*classmethod*`:** A class method is a method that belongs to a class and has access to class variables. It takes a reference to the class

as its first argument and is defined using the `@classmethod` decorator.

- **Instance method:** An instance method is a method that belongs to an instance of a class and has access to instance variables. It takes a reference to the instance as its first argument (usually named `self`).

34.What is the difference between `__new__` and `__init__` in Python?

`__*new*__` and `__*init*__` are special methods in Python that are involved in the object creation process. `__new__` is responsible for creating and returning a new instance of the class, while `__*init*__` is responsible for initializing the instance after it has been created. At the beginning the `__new__` method is called and then `__init__` is called. In most cases, you only need to override `__*init*__`.

35.What is the purpose of the `__call__` method in Python?

The `__*call*__` method is a special method in Python that allows an object to be called as a function. When an object is called as a function, the `__*call*__` method is executed. This can be useful for creating objects that behave like functions, such as decorators or function factories.

36. What is the difference between `*iter()*` and `*next()*` functions in Python?

The `*iter()*` function is used to obtain an iterator from an iterable object, while the `*next()*` function retrieves the next item from an iterator. When the iterator is exhausted, the `*next()*` function raises a `StopIteration` exception.

37. What is the purpose of the `*collections*` module in Python?

The `*collections*` module provides specialized container datatypes that can be used as alternatives to the built-in containers (list, tuple, dict, and set). Some of the most commonly used classes in the `collections` module are `*namedtuple*`, `*deque*`, `*Counter*`, `*OrderedDict*`, and `*defaultdict*`.

38. What is the purpose of the `*functools*` module in Python?

The `*functools*` module provides higher-order functions and tools for working with functions and other callable objects. Some of the most commonly used functions in the `functools` module are `partial`, `reduce`, `lru_cache`, `total_ordering`, and `wraps`.

39. What is the purpose of the `*itertools*` module in Python?

The `*itertools*` module provides a collection of fast, memory-efficient tools for working with iterators. Some of the most commonly used functions in the `*itertools*` module are `*count*`, `*cycle*`, `*repeat*`, `*chain*`, `*compress*`, `*dropwhile*`, `*takewhile*`, `*groupby*`, and `*zip_longest*`.

40.What is the purpose of the `*os*` and `*sys*` modules in Python?

The `*os*` module provides a way to interact with the operating system, such as file and directory management, process management, and environment variables. The `*sys*` module provides access to Python's runtime environment, such as command-line arguments, the Python path, and the standard input, output, and error streams.

41.What is the purpose of the `*threading*` and `*multiprocessing*` modules in Python?

The `*threading*` module provides a way to create and manage threads in Python, allowing you to write concurrent programs. The `*multiprocessing*` module provides a way to create and manage processes in Python, allowing you to write parallel programs that can take advantage of multiple CPU cores.

42.What are Python generators?

- Python generators are a type of iterable, like lists or tuples, but they lazily generate values on the fly without storing them in memory.
- This makes generators highly efficient for working with large datasets or infinite sequences, as they produce items one at a time and only as needed.
- Generators are created using functions and the yield keyword. Instead of returning a value and exiting, a generator function automatically suspends and resumes its execution and state around the last point of value generation.
- The advantage of generators over lists is their ability to generate a sequence of values over time rather than computing the sequence all at once and holding it in memory

43. How do you use Python generators?

- To use Python generators, first, we define a generator function by using the “def” keyword and “yield” instead of “return” to provide a value to the caller and pause execution
- We can iterate over a generator by using a “for-loop” or the “next()” function for each iteration to produce the next value in the sequence.

44. How can you achieve concurrency in Python?

- In Python, concurrency can be achieved through multiple mechanisms, each suited to different scenarios.
- For example, for I/O-bound tasks where the program waits for external events, we can use threading, which utilizes threads to execute multiple threads concurrently in a single process.
- For such tasks, we can also use Asynchronous Programming (asyncio), which allows a single thread to handle multiple tasks by waiting for I/O operations to complete without blocking.
- For CPU-bound tasks that require parallel computation, we can use “Multiprocessing” to leverage multiple processes instead of threads, each with its own Python interpreter and memory space, thus bypassing the Global Interpreter Lock (GIL).
- We can also use “Concurrent Futures,” which is a high-level interface for asynchronously executing callables using threads or processes to avoid the direct management of threads and processes for simple execution of functions concurrently.

45.Describe how you handle exceptions in Python.

- In Python, we do exception handling by the use of “try-except” blocks to allow a program to handle errors and continue its execution or provide informative feedback to the user.
- The “try” Block encapsulates the code that might raise an exception. So. if we suspect a code to cause an error, we put it in the “try” block.
- We follow the “try” block with one or more “except” blocks to catch and handle specific exceptions. We can specify the type of exception to catch (e.g., “ValueError“, “TypeError“), or use a generic “except” to catch any exceptions.
- As an option, after the “except” blocks, we can include an “else” block that runs if the code in the “try” block did not raise an exception.
- Another option is to use a “finally” block to define cleanup actions that must be executed under all circumstances, such as closing files or releasing resources, regardless of whether an exception was raised.

46.Explain Python’s use of the “with” statement and context managers.

- When used with context managers, the “with” statement in Python automatically manages the setup and teardown of resources. The “_enter_” method is called at the beginning of the block, initializing the context, and the “_exit_” method is invoked at the end, cleaning up the resources, regardless of whether the block was exited normally or due to an exception
- For example, when working with files, the “with” statement ensures that the file is properly closed after its block is executed, even if exceptions are raised within the block

47.Is Dictionary Lookup Faster than List Lookup? Why?

- Dictionary lookup time is generally faster, with a complexity of O(1), due to their hash table implementation. In contrast, list lookup time is O(n), where the entire list may need to be iterated to find a value.

48.What is the purpose of the 'finally' block?

- The `'finally'` block defines a block of code that will be executed regardless of whether an exception is raised or not.

49.What is the difference between 'xrange' and 'range' functions?

`'Range()'` and `'xrange()'` are both used for looping, but `'xrange()'` was available in Python 2 and behaves similarly to `'range()'` in Python 3. `'Xrange()'` is generated only when required, leading to its designation as "lazy evaluation."

50. Is indentation required in Python?

Indentation is absolutely essential in Python. It not only enhances code readability but also defines code blocks. Proper indentation is crucial for correct code execution; otherwise, you are left with a code that is not indented and difficult to read.

51.What are global, protected, and private attributes in Python?

The attributes of a class are also called variables. There are three access modifiers in Python for variables, namely

a. public – The variables declared as public are accessible everywhere, inside or outside the class.

b. private – The variables declared as private are accessible only within the current class.

c. protected – The variables declared as protected are accessible only within the current package.

Attributes are also classified as:

– Local attributes are defined within a code-block/method and can be accessed only within that code-block/method.

– Global attributes are defined outside the code-block/method and can be accessible everywhere.

```
class Mobile:
    m1 = "Samsung Mobiles" //Global attributes
    def price(self):
        m2 = "Costly mobiles"   //Local attributes
        return m2
Sam_m = Mobile()
print(Sam_m.m1)
```

52 . What is the difference between .py and .pyc files?

- **.py files are Python source files whereas .pyc files are compiled bytecode files generated by the Python compiler.**
- **Before executing a python program python interpreter checks for the compiled files. If the file is present, the virtual machine executes it. If not found, it checks for a .py file. If found, compile it to a .pyc file, and then the python virtual machine executes it.**
- **Having a .pyc file saves you the compilation time.**

53.What are the limitations of Python?

There are limitations to Python, which include the following

- It has design restrictions.
- It is slower when compared with C and C++ or Java.
- It is inefficient for mobile computing.
- It consists of an underdeveloped database access layer.

54.What are the different stages of the life cycle of a thread?

The different stages of the life cycle of a thread are:

- Stage 1: Creating a class where we can override the run method of the Thread class.
- Stage 2: We make a call to start() on the new thread. The thread is taken forward for scheduling purposes.
- Stage 3: Execution takes place wherein the thread starts execution and it reaches the running state.
- Stage 4: Thread waits until the calls to methods, including join() and sleep(), take place.
- Stage 5: After the waiting or execution of the thread, the waiting thread is sent for scheduling.
- Stage 6: Running thread is done by executing the terminates and reaches the dead state.

55.Draw a comparison between the range and xrange in Python.

- In terms of functionality, both range and xrange are identical. Both allow for generating a list of integers. The main difference between the two is that while range returns a Python list object, xrange returns an xrange object.
- Xrange is not able to generate a static list at runtime the way range does. On the contrary, it creates values along with the requirements via a special technique called yielding. It is used with a type of object known as a generator.
- If you have an enormous range for which you need to generate a list, then xrange is the function to opt for. This is especially

relevant for scenarios dealing with a memory-sensitive system, such as a smartphone.
- The range is a memory hog. Using it requires much more memory, especially if the requirement is gigantic. Hence, in creating an array of integers to suit the needs, it can result in a memory error and ultimately lead to a crash.

56.What are Python modules?

A file containing Python code, like functions and variables, is a Python module. A Python module is an executable file with a .py extension. Some built-in modules are:

- os
- sys
- math
- random
- data time
- JSON

57.What are ODBC modules in Python?

- The Microsoft Open Database Connectivity is an interface for the C programming language. It is the standard for all APIs using database C. If you use a Python ODBC interface with the standard ODBC drivers that ship with most databases, you can likely connect your Python application with most databases in the market. The different Python ODBC modules are pyodbc, PythonWin ODBC, and MxODBC.

Django

58.Can you tell me about Django and how it’s used by Python developers?

Django is a powerful Python web framework that helps developers create robust, scalable, and maintainable web applications. It offers a suite of tools, conventions, and libraries to help developers work efficiently and focus on application-specific code.

Some of the key features of Django include:

Simplified form handling

Object-relational mapping (ORM)

URL routing and views

A smooth, user-friendly interface for managing application data

User authentication and permission management

Advanced built-in security

Python developers can use Django to create different types of web apps, including content management systems (CMS), e-commerce websites, APIs, social media platforms, and more.

59.What's the difference between a Python package and a Python module?

- Packages and modules are both mechanisms for organizing and structuring code, but they have different purposes and characteristics.
- For starters, a Python module is a single file containing Python code. It can define functions, variables, and other objects that are used elsewhere in your program. Because of this, modules are particularly useful for organizing related code into separate files, enabling you to easily manage your codebase and improve code reuse.
- Meanwhile, packages are code packets that contain multiple modules and/or sub-packages. This enables you to organize related modules in a single directory.
- Packages are particularly important for larger projects that involve multiple code files and functionalities.

60.How would you use Python to fetch every 10th item from a list?

The easiest way to fetch every 10th item from a list is with a technique called "slicing". Here's an example of how you do it:

```
original_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
# Fetch every 10th item using slice notation
every_10th_item = original_list[::10]
print(every_10th_item)
```

This example would return 0, 10, and 20. If you want to start from a different number, you can modify the every_10th_item = original_list[::10] line.

```
every_10th_item_starting_from_index_2 = original_list[2::10]
```

print(every_10th_item_starting_from_index_2)
This example would return 2, 12.
Remember that Python is a zero-based language, which means that the first element is at index 0.

61.What are metaclasses in Python and why are they important?

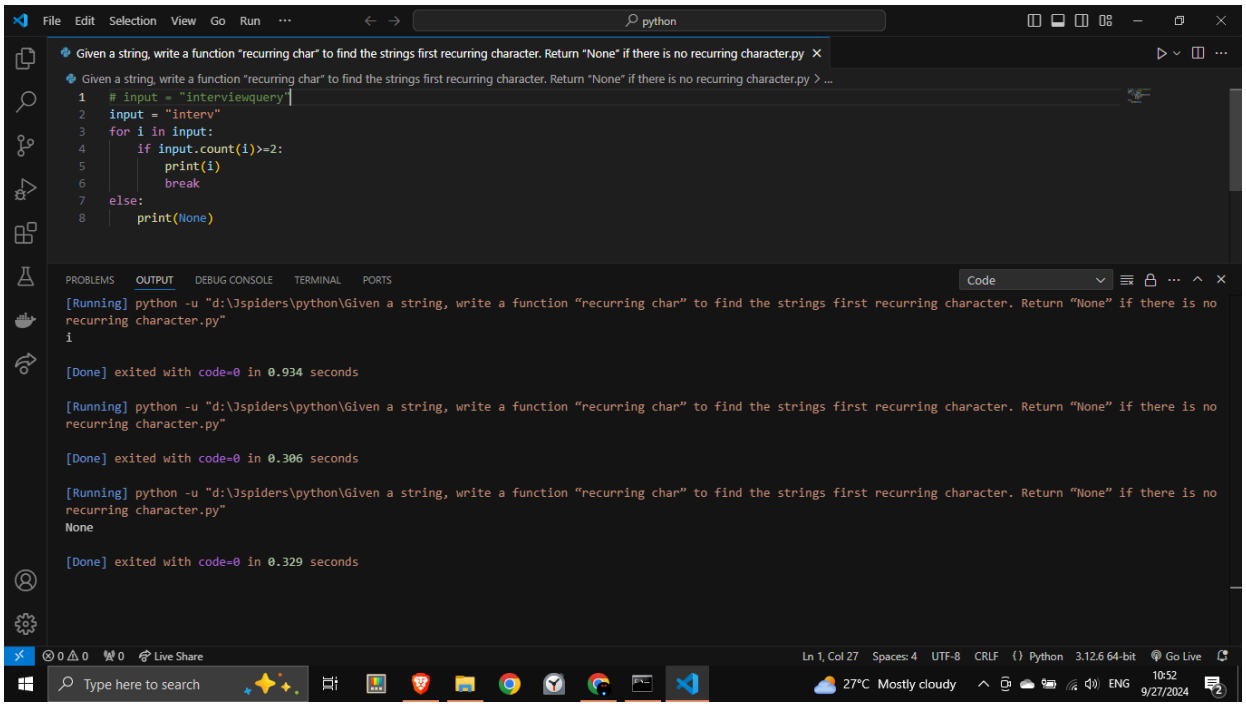
Metaclasses enable you to define the behavior of Python classes.
In simple terms, you can think of metaclass as a class for classes. They define how classes are created, how they interact, and what attributes they have.
Here are a few reasons why Python metaclasses are so important:

1. **Code reusability.** Since all classes within a metaclass are defined by the same behaviors, they contain a common logic. This makes it much easier to reuse code.
2. **Dynamically modifying classes**. With metaclasses, you can dynamically modify class attributes and methods when you're creating them, enabling dynamic code generation and automatic registration of subclasses, among other things.
3. **Customizing class creation.** This enables you to define the behavior of all classes created with this metaclass.
4. **Enforcing best practices.** With metaclasses, you can ensure that certain attributes are present or methods are defined in subclasses. This enables you to enforce design patterns or best practices in your code base.

62.What is faster for lookups in Python: dictionaries or lists?

With lists in Python, the time complexity is linear and is dependent on the number of values within the list. The lookup value is O(n). With dictionaries, the time complexity is constant because dictionaries are hash tables. You can find the value as O(1).
Because of this, dictionary lookups are generally faster in Python.

63. Given a string, write a function “recurring char” to find the strings first recurring character. Return “None” if there is no recurring character.



64.How can you sort elements in Python dictionary?

In the dictionary, you can easily sort the elements. For example, if we want to print the name of the elements of our dictionary alphabetically, we have to use for loop. It will sort each element of the dictionary accordingly.

```
Dict = {'Tim':
18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
Students = list(Dict.keys())
Students.sort()
for S in Students:
      print(":".join((S,str(Dict[S]))))
```

65 What are all dictionary methods:

Here is the list of dictionary methods:

- *copy()*
- *update()*
- *items()*
- *sort()*
- *len()*
- *cmp()*
- *Str()*

66.What are some built-in modules in Python?

Python modules are an essential concept around which you can expect tons of Python interview questions based on theory and problem-solving. Modules are files that contain Python code. Commonly used built-in modules in Python include:

- Sys
- Os
- Math
- Random
- JSON
- Data time

67.What is _init_ in Python?

init in Python is a constructor that is automatically called to allocate memory when a new object is created. All classes in Python have the _init_ function.

68.Does Python allow Multithreading?

Python has a multithreading package and allows multithreading through the Global Interpreter Lock construct. The construct allows only one thread to execute at one time and quickly moves onto the next thread for execution. The process is so quick that it may seem like multiple threads are executing simultaneously. Allowing threads to execute through this method is the most efficient way to run code.

69.Name some commonly-used libraries in Python

Libraries are essentially a collection of packages. Some popular Python libraries are Pandas, Matplotlib, Numpy, and Scikit-learn.

70.What do you understand about polymorphism in Python?

Polymorphism is a feature that allows methods to have multiple functionalities with the same name. For instance, if a parent class contains a method ABC, the child class can contain the same method with its own unique set of variables and functions.

71.Why is Python a widely used language?

Python is a high-end, general-purpose, interpreted programming language that allows developers to build complex software programs. With the help of the right tools and libraries, Python can be used to build a ton of applications and programs.

72.What exactly is a NumPy array?

NumPy arrays are much more versatile than Python lists. Reading and writing objects are quicker and more efficient using NumPy arrays.

73.In Python, how do you create random numbers?

We can create random data in Python utilizing several functions. They are as follows:

- **random() - This instruction gives a floating-point value ranging from 0 to 1.**
- **uniform(X, Y) - This function gives a floating-point value in the X and Y range.**
- **randint(X, Y) - This function gives a random integer between X and Y values.**

74.Is Python programming language or scripting language?

Python is a programming language that is frequently utilized for prearranging because of its straightforwardness and coherence. Be that as it may, it is a flexible language fit for taking care of a great many undertakings, from little scripts to huge scope programming improvement.

75.Python is an interpreted programming language. Explain.

Python is a deciphered programming language, significance its code is executed line by line by the Python translator without a different gathering step. This prompts more prominent adaptability, simplicity of improvement, and movability, though with possibly somewhat more slow execution contrasted with gathered dialects.

76.What's the distinction between .py and.pyc files?

The Python code we save is contained in the .py files. The .pyc files are created when the program is integrated into the current program from some other source. This file contains the bytecode for the Python files that we imported. The interpreter reduces processing time if we transform the source files having the format of .py files to .pyc files.

77.Explain all string operators with example

String operators with example:

Operator	Description	Example
[]	Slice- it gives the letter from the given index	a[1] will give “u” from the word Guru as such (0=G, 1=u, 2=r and 3=u)
[:]	Range slice-it gives the characters from the given range	x [1:3] it will give “ur” from the word Guru. Remember it will not consider 0, which is G, it will consider word after that is ur.

in	Membership-returns true if a letter exists in the given string	u is present in word Guru, and hence it will give 1 (True)
not in	Membership-returns true if a letter exists is not in the given string	l not present in word Guru and hence it will give 1
r/R	Raw string suppresses the actual meaning of escape characters.	Print r'\n' prints \n and print R'/n' prints \n
% – Used for string format	%r – It insert the canonical string representation of the object (i.e., repr(o)) %s- It inserts the presentation string representation of the object (i.e., str(o)) %d- it will format a number for display	The output of this code will be "guru 99".
+	It concatenates 2 strings	It concatenates strings and gives the result

*	Repeat	It prints the character twice.

78.Explain the split(), sub(), and subn() methods of the Python "re" module.

Python's "re" module provides three ways for modifying strings. They are as follows:

split() "splits" a string into a list using a regex pattern.

sub() finds all substrings that match the regex pattern given by us. Then it replaces that substring with the string provided.

subn() is analogous to sub() in that it gives the new string and the number of replacements.

79.Can Python be used for web client and web server side programming? Which one is best suited to Python?

Python is best suited for web server-side application development due to its vast set of features for creating business logic, database interactions, web server hosting, etc.

However, Python can be used as a web client-side application that needs some conversions for a browser to interpret the client-side logic.

Note: Python can be used to create desktop applications that can run as a standalone application such as utilities for test automation.

80.Mention at least 3-4 benefits of using Python over the other scripting languages such as Javascript.

Enlisted below are some of the benefits of using Python:

- Application development is faster and easy.
- Extensive support of modules for any kind of application development, including data analytics/machine learning/math-intensive applications.
- Community is always active to resolve user's queries.

81.Name some of the important modules that are available in Python.

Networking, Mathematics, Cryptographic services, Internet data handling, and Multi-threading modules are prominent modules. Apart from these, there are several other modules that are available in the Python developer community.

82. As Python is more suitable for server-side applications, it is very important to have threading implemented in your server code. How can you achieve that in Python?

We should use the threading module to implement, control, and destroy threads for parallel execution of the server code. Locks and Semaphores are available as synchronization objects to manage data between different threads.

83.Do we need to call the explicit methods to destroy the memory allocated in Python?

In Python “Garbage” collection is an in-built feature that takes care of allocating and deallocating memory. This is very similar to the feature in Java. Hence, there are very fewer chances of memory leaks in your application code.

84.What are help () and dir() in python?

“help ()” is a built-in function that can be used to return the Python documentation of a particular object, method, attributes, etc.

“dir ()” displays a list of attributes for the objects which are passed as an argument. If dir() is without the argument then it returns a list of names in the current local space.

85.What is the use of Assertions in Python?

Assert statement is used to evaluate the expression attached. If the expression is false, then python raises an AssertionError Exception.

86. What are the generators in python?

Generators are the functions in Python which return the repeated sets of items. They are used to create iterators. In Python, the “yield” keyword is used to create the generators.

Example:

```
def demo_gen():
    num = 10
    print('Statement number 1')
    yield num
    num += 1
    print('Statement number 2')
    yield num
    num += 1
    print('Statement number 3')
    yield num
for item in demo_gen():
    print(item)
```

Output:

Statement number 1

1

Statement number 2

2

Statement number 3

3

87.How is Multithreading achieved in Python?

Multithreading in Python is achieved using the threading module, which allows you to run multiple threads (smaller units of a process) concurrently. However, due to the Global Interpreter Lock (GIL), Python threads do not run in true parallelism for CPU-bound tasks. Still, they can be helpful in I/O-bound operations like file handling, network requests, or database queries.

Here's how you can achieve multithreading in Python:

a. Using the Threading Module

The threading module provides a way to create and manage threads. You can create a new thread by defining a target function and starting the thread.

```
import threading
# Define a function to run in a separate thread
def print_numbers():
    for i in range(5):
        print(i)
# Create a thread
```

```
thread = threading.Thread(target=print_numbers)
# Start the thread
thread.start()
# Wait for the thread to finish
thread.join()
print("Thread has completed.")
```

b. Using Subclassing

You can also subclass the Thread class to create your own threads.

```
import threading

class MyThread(threading.Thread):

    def run(self):

        for i in range(5):

            print(i)

# Create a new thread object

thread = MyThread()

# Start the thread

thread.start()
```

```
# Wait for the thread to complete

thread.join()

print("Custom thread has completed.")
```

c. Thread Synchronization

To prevent race conditions, Python provides synchronization primitives like locks (threading.Lock()), which ensure that only one thread accesses a shared resource at a time.

```
import threading

lock = threading.Lock()

shared_variable = 0

def increment():

    global shared_variable

    with lock:  # Acquire lock to prevent race condition

        shared_variable += 1

# Create multiple threads

threads = [threading.Thread(target=increment) for _ in range(10)]
```

```
# Start threads
for thread in threads:
    thread.start()
# Wait for all threads to finish
for thread in threads:
    thread.join()
print("Final shared variable:", shared_variable)
```

88.How do you display the contents of a text file in reverse order?

To display the contents of a text file in reverse order in Python, you can read the file and then reverse the lines or characters. Here's how you can do it:

Reverse Lines

```
with open('filename.txt', 'r') as file:
    lines = file.readlines()
    for line in reversed(lines):
        print(line.strip())
```

Reverse Characters

```
with open('filename.txt', 'r') as file:
    content = file.read()
    print(content[::-1])
```

- The first example reverses the lines of the file.
- The second example reverses the entire content of the file, character by character.

89. How would you obtain the res_set from the train_set and the test_set from below?

To split a dataset into train_set and test_set, you can use scikit-learn's train_test_split() function. This function will randomly divide your dataset into training and testing subsets.

```
from sklearn.model_selection import train_test_split

# Assuming res_set is your dataset

res_set = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Split res_set into 80% train_set and 20% test_set
```

```
train_set, test_set = train_test_split(res_set, test_size=0.2,
random_state=42)

print("Train Set:", train_set)

print("Test Set:", test_set)
```

Here, 80% of the data will be in train_set and 20% in test_set. You can adjust test_size for different splits.

90. Differentiate between .pyc and .py.

The .py files are the source code for Python. The bytecode is stored in .pyc files, which are created when code is imported from another source. The interpreter saves time by converting the source .py files to .pyc files.

91.What are Python iterators?

In Python, an iterator is an object that allows you to traverse through a collection's elements (such as lists, tuples, or dictionaries) one at a time. An iterator implements two methods: __iter__() (which returns the iterator object itself) and __next__() (which returns the next item in the sequence and raises StopIteration when the sequence is exhausted).

92.What are generators in Python?

Generators in Python are a special type of function that return an iterator and allow you to iterate through a sequence of values lazily, meaning they produce values one at a time and only when requested. Instead of using return, generators use the yield keyword to return values. This makes generators memory-efficient, especially for large datasets, as they generate values on the fly rather than storing the entire sequence in memory.

```
def my_generator():
    yield 1
    yield 2
    yield 3

gen = my_generator()

for value in gen:
    print(value)
```

Output

```
1
2
3
```

93.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**

94.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
```

95. How will you check if a class is a child of another class?

This is done by using a method called issubclass() provided by python. The method tells us if any class is a child of another class by returning true or false accordingly.
For example:

```
class Parent(object):
   pass

class Child(Parent):
   pass

# Driver Code
```

```
print(issubclass(Child, Parent))    #True
print(issubclass(Parent, Child))    #False
```

- We can check if an object is an instance of a class by making use of isinstance() method:

```
obj1 = Child()
obj2 = Parent()
print(isinstance(obj2, Child))    #False
print(isinstance(obj2, Parent))   #True
```

96.Why is finalize used?

Finalize method is used for freeing up the unmanaged resources and cleaning up before the garbage collection method is invoked. This helps in performing memory management tasks.

97. Differentiate between new and override modifiers.

The new modifier is used to instruct the compiler to use the new implementation and not the base class function. The Override modifier is useful for overriding a base class function inside the child class.

98. Differentiate between deep and shallow copies.

- Shallow copy does the task of creating new objects storing references of original elements. This does not undergo recursion to create copies of nested objects. It just copies the reference details of nested objects.

- Deep copy creates an independent and new copy of an object and even copies all the nested objects of the original element recursively.

99.What is main function in python? How do you invoke it?

In the world of programming languages, the main is considered as an entry point of execution for a program. But in python, it is known that the interpreter serially interprets the file line-by-line. This means that python does not provide main() function explicitly. But this doesn't mean that we cannot simulate the execution of main. This can be done by defining user-defined main() function and by using the __name__ property of python file. This __name__ variable is a special built-in variable that points to the name of the current module. This can be done as shown below:

```
def main():
  print("Hi Interviewbit!")
if __name__=="__main__":
  main()
```

100.Are there any tools for identifying bugs and performing static analysis in python?

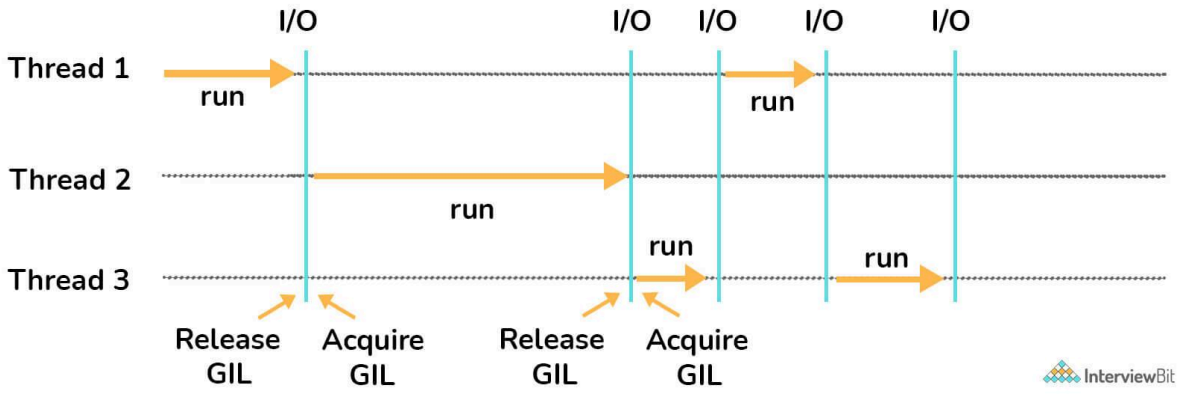
Yes, there are tools like PyChecker and Pylint which are used as static analysis and linting tools respectively. PyChecker helps find bugs in python source code files and raises alerts for code issues and their complexity. Pylint checks for the module's coding standards and supports different plugins to enable custom features to meet this requirement.

101.Define PYTHONPATH.

It is an environment variable used for incorporating additional directories during the import of a module or a package. PYTHONPATH is used for checking if the imported packages or modules are available in the existing directories. Not just that, the interpreter uses this environment variable to identify which module needs to be loaded.

102.Define GIL.

GIL stands for Global Interpreter Lock. This is a mutex used for limiting access to python objects and aids in effective thread synchronization by avoiding deadlocks. GIL helps in achieving multitasking (and not parallel computing). The following diagram represents how GIL works.

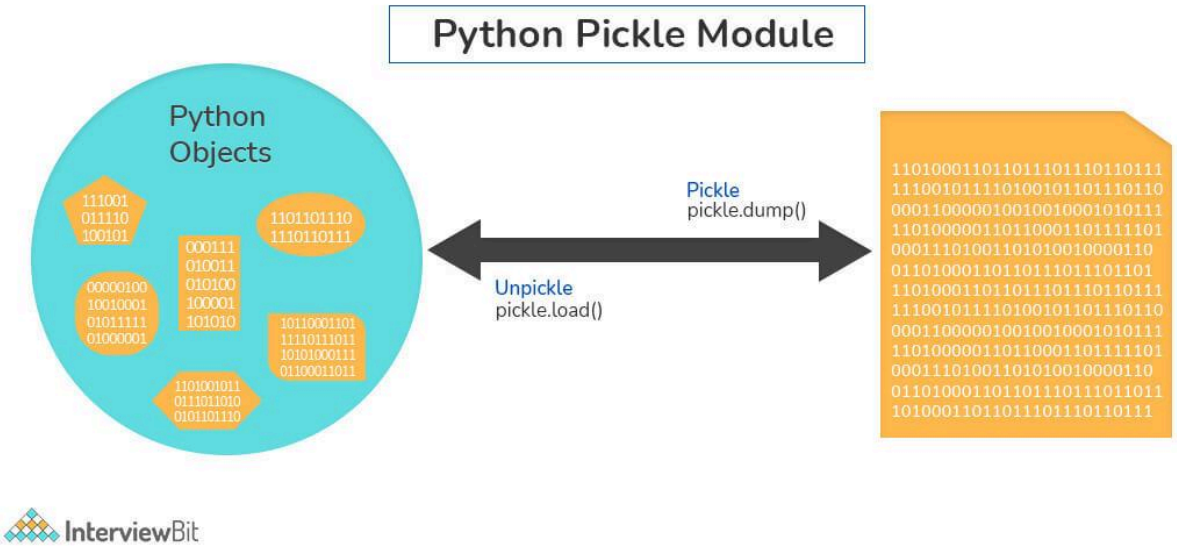


Based on the above diagram, there are three threads. First Thread acquires the GIL first and starts the I/O execution. When the I/O operations are done, thread 1 releases the acquired GIL which is then taken up by the second thread. The process repeats and the GIL are used by different threads alternatively until the threads have completed their execution. The threads not having the GIL lock goes into the waiting state and resumes execution only when it acquires the lock.

103.What are the differences between pickling and unpickling?

Pickling is the conversion of python objects to binary form. Whereas, unpickling is the conversion of binary form data to python objects. The pickled objects are used for storing in disks or external memory locations. Unpickled objects are used for getting the data back as python objects upon which processing can be done in python.

Python provides a pickle module for achieving this. Pickling uses the pickle.dump() method to dump python objects into disks. Unpickling uses the pickle.load() method to get back the data as python objects.



104.What are lambda functions?

Lambda functions are generally inline, anonymous functions represented by a single expression. They are used for creating function objects during runtime. They can accept any number of parameters. They are usually used where functions are required only for a short period. They can be used as:

```
mul_func = lambda x,y : x*y

print(mul_func(6, 4))

# Output: 24
```

105.Differentiate between a package and a module in python.

The module is a single python file. A module can import other modules (other python files) as objects. Whereas, a package is the folder/directory where different sub-packages and the modules reside.
A python module is created by saving a file with the extension of .py. This file will have classes and functions that are reusable in the code as well as across modules.
A python package is created by following the below steps:

- Create a directory and give a valid name that represents its operation.
- Place modules of one kind in this directory.
- Create __init__.py file in this directory. This lets python know the directory we created is a package. The contents of this package can be imported across different modules in other packages to reuse the functionality.

106.Write a Program to convert date from yyyy-mm-dd format to dd-mm-yyyy format.

We can again use the re module to convert the date string as shown below:
import re

```
def transform_date_format(date):
   return re.sub(r'(\d{4})-(\d{1,2})-(\d{1,2})', '\\3-\\2-\\1', date)
date_input = "2021-08-01"
print(transform_date_format(date_input))
```

You can also use the datetime module as shown below:

```
from datetime import datetime
new_date = datetime.strptime("2021-08-01",
"%Y-%m-%d").strftime("%d:%m:%Y")
print(new_data)
```

107.Write a Program to add two integers >0 without using the plus operator.

We can use bitwise operators to achieve this.

```
def add_nums(num1, num2):
  while num2 != 0:
     data = num1 & num2
     num1 = num1 ^ num2
     num2 = data << 1
  return num1
print(add_nums(2, 10))
```

108.Write a program for counting the number of every character of a given text file.

The idea is to use collections and pprint module as shown below:

```
import collections
import pprint
with open("sample_file.txt", 'r') as data:
 count_data = collections.Counter(data.read().upper())
 count_value = pprint.pformat(count_data)
print(count_value)
```

109.Write a Program to match a string that has the letter ‘a’ followed by 4 to 8 'b’s.

We can use the re module of python to perform regex pattern comparison here.

```
import re

def match_string(text):
    # Regular expression to match 'a' followed by 4 to 8 'b's
    pattern = r"ab{4,8}"
    if re.fullmatch(pattern, text):
        return "Match found!"
    else:
        return "No match."

# Test cases
print(match_string("abbbb"))     # Match found
print(match_string("abbbbbbb"))  # Match found
print(match_string("abbbbbbbb")) # Match found
print(match_string("abbb"))      # No match
print(match_string("abbbbbbbbb")) # No match
```

- The pattern `ab{4,8}` matches a string starting with 'a' followed by 4 to 8 occurrences of 'b'.
- `re.fullmatch()` is used to ensure the entire string matches the pattern exactly.

This will return "Match found!" only for strings with 'a' followed by 4 to 8 'b's.

110.How is Multithreading achieved in Python?

Multithreading in Python is achieved using the threading module, which allows you to run multiple threads (smaller units of a process) concurrently. However, due to the Global Interpreter Lock (GIL), Python threads do not run in true parallelism for CPU-bound tasks. Still, they can be helpful in I/O-bound operations like file handling, network requests, or database queries.

Here's how you can achieve multithreading in Python:

a. Using the Threading Module

The threading module provides a way to create and manage threads. You can create a new thread by defining a target function and starting the thread.

```
import threading
# Define a function to run in a separate thread
def print_numbers():
    for i in range(5):
```

```
        print(i)
# Create a thread
thread = threading.Thread(target=print_numbers)
# Start the thread
thread.start()
# Wait for the thread to finish
thread.join()
print("Thread has completed.")
```

b. Using Subclassing

You can also subclass the Thread class to create your own threads.

```
import threading
class MyThread(threading.Thread):
    def run(self):
        for i in range(5):
            print(i)
# Create a new thread object
thread = MyThread()
# Start the thread
thread.start()
# Wait for the thread to complete
thread.join()
print("Custom thread has completed.")
```

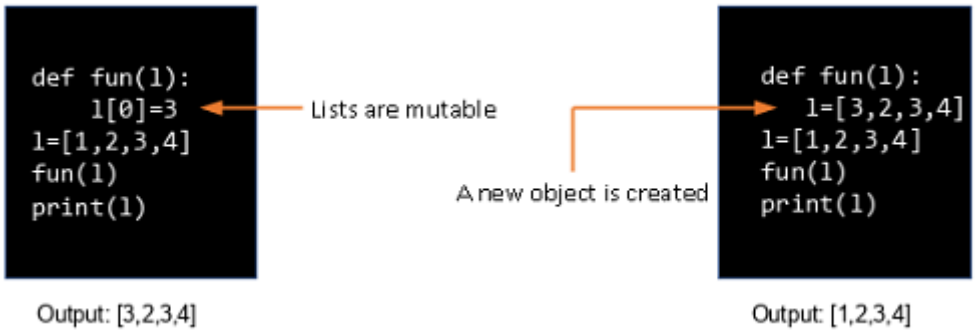
c. Thread Synchronization

To prevent race conditions, Python provides synchronization primitives like locks (threading.Lock()), which ensure that only one thread accesses a shared resource at a time.

```
import threading
lock = threading.Lock()
shared_variable = 0
def increment():
    global shared_variable
    with lock:  # Acquire lock to prevent race condition
        shared_variable += 1
# Create multiple threads
threads = [threading.Thread(target=increment) for _ in range(10)]
# Start threads
for thread in threads:
    thread.start()
# Wait for all threads to finish
for thread in threads:
    thread.join()
print("Final shared variable:", shared_variable)
```

111.Are arguments in Python passed by value or by reference?

Python passes arguments by reference. Any changes made within a function are reflected in the original object.
Consider two sets of code shown below:



In the first example, we only assigned a value to one element of 'l', so the output is [3, 2, 3, 4].
In the second example, we have created a whole new object for 'l'. But the values [3, 2, 3, 4] don't appear in the output as they are outside the function's definition.

112.In Python, functions are first-class objects." What do you infer from this?

The statement "In Python, functions are first-class objects" means that functions in Python are treated like any other object. You can:

- Assign functions to variables
    - Example: f = my_function
- Pass functions as arguments to other functions
    - Example: some_function(my_function)
- Return functions from other functions
    - Example: return my_function

- Store functions in data structures
- Example: functions_list = [func1, func2]

This flexibility allows functions to be used in more dynamic and powerful ways, such as in higher-order functions and callbacks.

113.What is the output of: Print(__name__)? Justify your answer.

The output of print(__name__) depends on where the code is executed:

- When the script is run directly: The output will be __main__. This is because when a Python script is executed, the __name__ variable is set to __main__, indicating that the script is running as the main program.
- When the script is imported as a module: The output will be the module's name (i.e., the filename without the .py extension). This helps differentiate between code that runs directly and code that is imported.

Example

print(__name__)

If run directly, the output will be:

__main__

114.What is a NumPy array?

A NumPy array is a powerful, grid-like data structure provided by the NumPy library in Python. It is designed to handle large, multi-dimensional arrays and matrices efficiently. Unlike Python lists, NumPy arrays are optimized for numerical computations, offering faster processing, memory efficiency, and support for vectorized operations. NumPy arrays can store elements of the same data type, and they are commonly used in scientific computing, data analysis, and machine learning.

```
import numpy as np
arr = np.array([1, 2, 3, 4])  # Creates a 1D NumPy array
```

115.What is the difference between Matrices and Arrays?

Matrices	Arrays
• A matrix comes from linear algebra and is a two-dimensional representation of data. • It comes with a powerful set of mathematical operations that allow you to manipulate the data in interesting ways.	• An array is a sequence of objects of similar data type. • An array within another array forms a matrix.

116.What is the difference between range() and xrange() functions in Python?

In Python 2, range() and xrange() both generate sequences of numbers, but they differ in how they handle memory:

- range(): Returns a list of numbers, storing all the values in memory at once. This can be inefficient for large ranges.
- xrange(): Returns an iterator that generates numbers on the fly, using less memory. It is more efficient for large ranges.

In Python 3, xrange() was removed, and range() now behaves like xrange(), generating numbers lazily without consuming extra memory.

117.What are the essential features of Python?

- Python is a scripting language. Unlike other programming languages like C and its derivatives, it does not require compilation prior to execution.
- Python is dynamically typed, which means you don't have to specify the kinds of variables when declaring them or anything.
- Python is well suited to object-oriented programming since it supports class definition, composition, and inheritance.

118.Differentiate between .pyc and .py.

The .py files are the source code for Python. The bytecode is stored in .pyc files, which are created when code is imported from another source. The interpreter saves time by converting the source .py files to .pyc files.

119.What is slicing in Python?

Slicing in Python is a technique used to extract a portion (or "slice") of a sequence like a list, tuple, or string. You can specify a range of indices to access elements in a sequence. The syntax is:
sequence[start:stop:step]

- start: The index where the slice starts (inclusive).
- stop: The index where the slice ends (exclusive).
- step (optional): Defines the step size or how many elements to skip.

```
my_list = [1, 2, 3, 4, 5]
print(my_list[1:4])  # Output: [2, 3, 4]
```

120.Why Lambda is used in Python?

Lambda is typically utilized when an anonymous function is required for a short period. Lambda functions can be applied in two different ways:

- Assigning Lambda functions to a variable
- Wrapping Lambda function into another function

121.Explain the split(), sub(), and subn() methods of the Python "re" module.

Python's "re" module provides three ways to modify strings. They are:
split (): a regex pattern is used to "separate" a string into a list
subn(): It works similarly to sub(), returning the new string as well as the number of replacements.
sub(): identifies all substrings that match the regex pattern and replaces them with a new string

122.Explain Python packages.

A Python package is a collection of modules grouped in a directory, typically to organize and manage related functionalities. Each package contains an __init__.py file, which indicates to Python that the directory should be treated as a package. Packages allow for better code organization and reusability by logically grouping related code components, such as functions, classes, or variables, into separate modules.

```
from mypackage import module1, module2
```

123.What is Flask and explain its benefits.

Flask is a Python web microframework based on the BSD license. Two of its dependencies are Werkzeug and Jinja2, which means it will have few if any, external library dependencies. This lightens the framework while reducing update dependencies and security vulnerabilities.
A session is remembering information from one request to the next. A session in a flask employs a signed cookie to allow the user to inspect and edit its contents. If the user only has the secret key, he or she can change the session. Flask.secret key.

124.Is Django better as compared to Flask?

Django and Flask map URLs or addresses entered into web browsers into Python functions.
Flask is easier to use than Django, but it doesn't do much for you, so you will have to specify the specifics, whereas Django does a lot for you, and you won't have to do anything. Django has prewritten

code that the user must examine, whereas Flask allows users to write their code, making it easier to grasp. Both are technically excellent and have their own set of advantages and disadvantages.

125.Differentiate between Pyramid, Django, and Flask.

- Pyramid is designed for larger apps. It gives developers flexibility and allows them to utilize the appropriate project tools. The database, URL structure, templating style, and other options are all available to the developer. Pyramids can be easily customized.
- Flask is a "microframework" designed for small applications with straightforward needs. External libraries are required in a flask. The flask is now ready for use.
- Django, like Pyramid, may be used for larger applications. It has an ORM in it.

126.What is GIL?

GIL stands for Global Interpreter Lock. This mutex helps thread synchronization by preventing deadlocks by limiting access to Python objects. GIL assists with multitasking (not parallel computing).

127.List some popular applications of Python.

Python is widely used in web development (e.g., Django, Flask), data science (e.g., Pandas, NumPy), machine learning (e.g., TensorFlow, Scikit-learn), automation (e.g., scripts), artificial intelligence, and game development (e.g., Pygame)

128.Can you explain common searching and graph traversal algorithms in Python?

Python has a number of different powerful algorithms for searching and graph traversal, and each one deals with different data structures and solves different problems. I can them here:

- Binary Search: If you need to quickly find an item in a sorted list, binary search is your go-to. It works by repeatedly dividing the search range in half until the target is found.
- AVL Tree: An AVL tree keeps things balanced, which is a big advantage if you're frequently inserting or deleting items in a tree. This self-balancing binary search tree structure keeps searches fast by making sure the tree never gets too skewed.
- Breadth-First Search (BFS): BFS is all about exploring a graph level by level. It's especially useful if you're trying to find the shortest path in an unweighted graph since it checks all possible moves from each node before going deeper.
- Depth-First Search (DFS): DFS takes a different approach by exploring as far as it can down each branch before backtracking. It's great for tasks like maze-solving or tree traversal.
- A Algorithm*: The A* algorithm is a bit more advanced and combines the best of both BFS and DFS by using heuristics to find the shortest path efficiently. It's commonly used in pathfinding for maps and games.

129.What is a KeyError in Python, and how can you handle it?

A KeyError in Python occurs when you try to access a key that doesn't exist in a dictionary. This error is raised because Python expects every key you look up to be present in the dictionary, and when it isn't, it throws a KeyError.

For example, if you have a dictionary of student scores and try to access a student who isn't in the dictionary, you'll get a KeyError. To handle this error, you have a few options:

- Use the .get() method: This method returns None (or a specified default value) instead of throwing an error if the key isn't found.
- Use a try-except block: Wrapping your code in try-except allows you to catch the KeyError and handle it gracefully.
- Check for the key with in: You can check if a key exists in the dictionary using if key in dictionary before trying to access it.

130.How does Python handle memory management, and what role does garbage collection play?

Python manages memory allocation and deallocation automatically using a private heap, where all objects and data structures are stored. The memory management process is handled by Python's memory manager, which optimizes memory usage, and the garbage collector, which deals with unused or unreferenced objects to free up memory.

Garbage collection in Python uses reference counting as well as a cyclic garbage collector to detect and collect unused data. When an object has no more references, it becomes eligible for garbage collection. The gc module in Python allows you to interact with the garbage collector directly, providing functions to enable or disable garbage collection, as well as to perform manual collection.

131.What is the difference between shallow copy and deep copy in Python, and when would you use each?

In Python, shallow and deep copies are used to duplicate objects, but they handle nested structures differently.

- Shallow Copy: A shallow copy creates a new object but inserts references to the objects found in the original. So, if the original object contains other mutable objects (like lists within lists), the shallow copy will reference the same inner objects. This can lead to unexpected changes if you modify one of those inner objects in either the original or copied structure. You can create a shallow copy using the copy() method or the copy module's copy() function.
- Deep Copy: A deep copy creates a new object and recursively copies all objects found within the original. This means that even nested structures get duplicated, so changes in one copy don't affect the other. To create a deep copy, you can use the copy module's deepcopy() function.

Example Usage: A shallow copy is suitable when the object contains only immutable items or when you want changes in nested structures to reflect in both copies. A deep copy is ideal when working with complex, nested objects where you want a completely independent duplicate. Read our Python Copy List: What You Should Know tutorial to learn more. This tutorial includes a whole section on the difference between shallow copy and deep copy.

132.What is monkey patching in Python?

Monkey patching in Python is a dynamic technique that can change the behavior of the code at run-time. In short, you can modify a class or module at run-time.

Example:

Let's learn monkey patching with an example.

1. We have created a class monkey with a patch() function. We have also created a monk_p function outside the class.
2. We will now replace the patch with the monk_p function by assigning monkey.patch to monk_p.
3. In the end, we will test the modification by creating the object using the monkey class and running the patch() function.

Instead of displaying patch() is being called, it has displayed monk_p() is being called.

```
class monkey:
    def patch(self):
        print ("patch() is being called")

def monk_p(self):
    print ("monk_p() is being called")

# replacing address of "patch" with "monk_p"
monkey.patch = monk_p

obj = monkey()

obj.patch()
# monk_p() is being called
```

133.Why use else in try/except construct in Python?

try: and except: are commonly known for exceptional handling in Python, so where does else: come in handy? else: will be triggered when no exception is raised.

Example:

Let's learn more about else: with a couple of examples.

1. On the first try, we entered 2 as the numerator and d as the denominator. Which is incorrect, and except: was triggered with "Invalid input!".
2. On the second try, we entered 2 as the numerator and 1 as the denominator and got the result 2. No exception was raised, so it triggered the else: printing the message Division is successful.

```
try:
    num1 = int(input('Enter Numerator: '))
    num2 = int(input('Enter Denominator: '))
    division = num1/num2
    print(f'Result is: {division}')
except:
    print('Invalid input!')
else:
    print('Division is successful.')
```

```
## Try 1 ##
# Enter Numerator: 2
# Enter Denominator: d
# Invalid input!

## Try 2 ##
# Enter Numerator: 2
```

```
# Enter Denominator: 1
# Result is: 2.0
# Division is successful.
```

134.What are decorators in Python?

Decorators in Python are a design pattern that allows you to add new functionality to an existing object without modifying its structure. They are commonly used to extend the behavior of functions or methods.

Example:

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
# Output:
# Something is happening before the function is called.
# Hello!
# Something is happening after the function is called.
```

135.What are context managers in Python, and how are they implemented?

Context managers in Python are used to manage resources, ensuring that they are properly acquired and released. The most common use of context managers is the with statement.
Example:

```
class FileManager:
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode

    def __enter__(self):
        self.file = open(self.filename, self.mode)
        return self.file

    def __exit__(self, exc_type, exc_value, traceback):
        self.file.close()

with FileManager('test.txt', 'w') as f:
    f.write('Hello, world!')
```


In this example, the FileManager class is a context manager that ensures the file is properly closed after it is used within the with statement.