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.