**Q1. Is an assignment operator like += only for show? Is it possible that it would lead to faster results at the runtime?**

Ans-No, an assignment operator like += is not just for show; it can lead to faster results at runtime. The += operator is a shorthand for assigning a new value that is the sum of the current value and another value. It can be more efficient than writing out the longer form x = x + y, especially in certain contexts, such as when working with mutable data structures like lists or arrays.

**Q2. What is the smallest number of statements you'd have to write in most programming languages to replace the Python expression a, b = a + b, a?**

Ans- The Python expression a, b = a + b, a assigns the value of a + b to a, and the original value of a to b. In most programming languages, you can achieve the same result with three statements:

temp = a

a = a + b

b = temp

**Q3. In Python, what is the most effective way to set a list of 100 integers to 0?**

Ans- The most effective way to set a list of 100 integers to 0 in Python is to use a list comprehension or the built-in list() function.

Example: list = [0 for \_ in range(100)]

**Q4. What is the most effective way to initialise a list of 99 integers that repeats the sequence 1, 2, 3? S If necessary, show step-by-step instructions on how to accomplish this.**

Ans- The most effective way to initialize a list of 99 integers that repeats the sequence 1, 2, 3 is to use the modulo operator (%).Here's how to do it step by step in Python:

1.Create an empty list to store the integers:

list = [ ]

2.Use the range() function to create a sequence of numbers from 0 to 98:

for i in range(99):

3.Calculate the remainder when i is divided by 3 (the length of the repeating sequence):

4.Add 1 to the list if the remainder is 0, 2 if the remainder is 1, and 3 if the remainder is any other:

if remainder == 0:

my\_list.append(1)

elif remainder == 1:

my\_list.append(2)

else:

my\_list.append(3)

5.After the loop has finished executing, the list will contain 99 integers that repeat the sequence 1, 2, 3:

print(my\_list)

**Q5. If you're using IDLE to run a Python application, explain how to print a multidimensional list as efficiently?**

Ans-To print a multidimensional list efficiently in Python using IDLE, you can use nested loops to iterate over each element in the list. Here is an example code:-

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

# Iterate over each element in the list and print it

for row in list:

for item in row:

print(item, end=' ')

print()

**Q6. Is it possible to use list comprehension with a string? If so, how can you go about doing it?**

Ans-Yes, it is possible to use list comprehension with a string in Python. You can use list comprehension to create a new list by iterating over each character in the string and applying a condition or transformation to each character.

Here's an example of how to use list comprehension with a string to create a list of all the uppercase letters in the string:

my\_string = "Hello, World!"

upper\_list = [char for char in my\_string if char.isupper()]

print(upper\_list)

**Q7. From the command line, how do you get support with a user-written Python programme? Is this possible from inside IDLE?**

Ans- From the command line, you can get support with a user-written Python program by using the ‘–help’ or ‘-h’ command-line argument. This will display the help documentation for the program, which should provide information about how to use the program and what command-line arguments are available.

**Q8. Functions are said to be “first-class objects” in Python but not in most other languages, such as C++ or Java. What can you do in Python with a function (callable object) that you can't do in C or C++?**

Ans- In Python, functions are considered "first-class objects" because they can be assigned to variables, passed as arguments to other functions, and returned as values from functions. This means that you can treat a function just like any other data type, such as integers or strings.Here are some things that you can do with functions in Python that are not possible in C or C++:

1. Functions can be assigned to variables: In Python, you can assign a function to a variable just like you would assign an integer or string. This allows you to pass functions as arguments to other functions or return them as values from functions.
2. Functions can be passed as arguments to other functions: In Python, you can pass a function as an argument to another function. This is useful when you want to pass a specific behavior to a function, rather than just a value.
3. Functions can be returned as values from functions: In Python, you can return a function as a value from a function. This is useful when you want to dynamically generate a function based on certain conditions.
4. Functions can be defined inside other functions: In Python, you can define a function inside another function. This allows you to create a function that is only accessible within the scope of the outer function.
5. Functions can be used as elements of data structures: In Python, you can store functions as elements of data structures such as lists, dictionaries, or sets. This is useful when you want to dynamically call different functions based on certain conditions.

**Q9. How do you distinguish between a wrapper, a wrapped feature, and a decorator?**

Ans-Wrapper, wrapped feature, and decorator are all programming concepts that involve adding functionality to an existing piece of code. Here's how you can distinguish between them:

Wrapper: A wrapper is a function or class that "wraps" or encapsulates another function or class, adding some additional functionality. The wrapper takes the original function or class as an argument and returns a new function or class that includes the additional functionality. The original function or class remains unchanged. A common use case for a wrapper is to add logging, timing, or error handling to a function or class.

Wrapped feature: A wrapped feature refers to the original function or class that is being wrapped by a wrapper. The wrapped feature is the functionality that is being extended or modified by the wrapper. The wrapped feature is the core functionality that the wrapper is building upon.

Decorator: A decorator is a special type of wrapper that is used to modify the behaviour of a function or class. A decorator takes a function or class as an argument and returns a new function or class that has been modified in some way. Unlike a regular wrapper, a decorator is used by placing an "@" symbol before the decorator's name above the function or class that it is modifying. A decorator can be used to add caching, memoization, or validation to a function or class.

**Q10. If a function is a generator function, what does it return?**

Ans-A generator function is a special type of function in Python that uses the "yield" keyword to return an iterator object that can be used to generate a sequence of values. When a generator function is called, it returns a generator object, which is an iterator that can be used to iterate over the sequence of values generated by the function.

**Q11. What is the one improvement that must be made to a function in order for it to become a generator function in the Python language?**

Ans- The one improvement that must be made to a function in order for it to become a generator function in Python is to replace the **return** statements in the function with **yield** statements.

Using a generator function can be more memory-efficient than generating all the values at once and returning them in a list, especially when dealing with large datasets or infinite sequences.

**Q12. Identify at least one benefit of generators.**

Ans- One benefit of generators in Python is that they allow for lazy evaluation of sequences, which can be more memory-efficient and faster than generating all the values at once and returning them in a list.

With generators, values are generated one at a time as they are requested, rather than generating them all at once and storing them in memory. This can be particularly useful when dealing with large datasets or infinite sequences, where it may not be practical or even possible to generate all the values at once.