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

**ANSWER:** No, an assignment operator like += is not just for show; it can be faster than the equivalent operation using the standard assignment operator (=) in some cases.

The reason for this is that the += operator combines the operation of addition and assignment into a single operation. This can be more efficient than using the standard assignment operator followed by a separate addition operation because it avoids creating an unnecessary temporary object.

**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?**

**ANSWER:** some programming languages may support a swapping idiom to swap two variables without using a temporary variable. For example, in JavaScript, you can swap the values of a and b using the following code:

**[b, a] = [a + b, a];**

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

**ANSWER:** In Python, you can set a list of 100 integers to 0 using a list comprehension or a loop, like this:

Using a List Comprehension:

my\_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.**

**ANSWER:** One way to initialize a list of 99 integers that repeats the sequence 1, 2, 3 is to use a list comprehension. Here's how you can do it:

lst = [i for i in range(1, 100)]

for i in range(3, 100, 3):

lst[i-1:i+2] = [1, 2, 3]

The first line initializes a list of integers from 1 to 99 using a list comprehension.

The second line then replaces every third element (starting from the third element) with the sequence 1, 2, 3.

After the loop is finished, ‘lst’ will contain the desired sequence of 1, 2, 3 repeated 33 times.

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

**ANSWER:** To print a multidimensional list efficiently in IDLE, you can use a nested loop to iterate over the rows and columns of the list and print each element. For Example:

my\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

for row in my\_list:

for element in row:

print(element, end=" ")

print() # Move to the next line after printing each row

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

**ANSWER:** Yes, you can use list comprehension with a string in Python. In fact, list comprehension can be used with any iterable object, including strings.

For Example:

my\_string = "hello world"

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

print(my\_list)

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

**ANSWER:** From the command line, you can get support with a user-written Python program by using the --help or -h option followed by the name of your Python script.

As for IDLE, it provides a built-in help system that you can use to get support with your Python program. To access the help system in IDLE, you can click on the "Help" menu and select "Python Docs". This will open the Python documentation in your web browser, where you can search for information on Python functions, modules, and other topics.

**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++?**

**ANSWER:** Here are some things you can do with functions in Python that you can't do in C++ or Java:

* Assign functions to variables
* Define functions inside other functions
* Pass functions as arguments to other functions
* Return functions from other functions
* Use functions as decorators

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

**ANSWER:** A wrapper is a function that encapsulates or modifies another function, the wrapped feature is the original function that is being encapsulated or modified, and a decorator is a special kind of wrapper function that is used to modify the behavior of another function using the "@" syntax.

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

**ANSWER:** If a function is a generator function, it doesn't return a regular value like other functions. Instead, it returns an iterator object that generates a sequence of values when iterated over. The values are generated one at a time using the "yield" keyword instead of "return".

**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?**

**ANSWER:** The one improvement that must be made to a regular function in order for it to become a generator function in Python is to use the "yield" keyword instead of "return" to generate values.

When a regular function encounters a "return" statement, it terminates and returns a value to the caller. In contrast, when a generator function encounters a "yield" statement, it returns a value to the caller, but it also saves its state and remembers where it left off, so that it can resume execution from that point when the generator is iterated again.

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

**ANSWER:** One benefit of generators in Python is that they can generate large sequences of values on-the-fly without storing them all in memory at once. This can be especially useful when working with large data sets or when the memory is limited.

Another benefit of generators is that they can be composed and chained together using generator expressions or the built-in "zip" and "map" functions. This can make code more concise, readable, and efficient, and can allow for complex operations to be performed on large data sets without loading them all into memory at once.