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

No, the assignment operator like "+=" is not just for show. It is a shorthand notation for performing addition and assignment in a single step. It can lead to faster results at runtime because it avoids the need to explicitly write separate addition and assignment statements, reducing code length and potentially improving performance.

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?

In most programming languages, you would need a minimum of three statements to replace the Python expression "a, b = a + b, a". One possible approach could be:

1. Create a temporary variable to hold the sum of a and b.

2. Assign the value of a to b.

3. Assign the value of the temporary variable to a.

These three statements would be required to achieve the same result as the Python expression.

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

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.

my\_list = [(i % 3) + 1 for i in range(99)]

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

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

for sublist in my\_list:

for element in sublist:

print(element, end=" ")

print()

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

my\_string = "Hello World"

new\_list = [char for char in my\_string]

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

From the command line, you can get support with a user-written Python program by using the `python` command along with the `-h` or `--help` option to display the program's help information or usage instructions, if provided by the developer.

Inside IDLE, you can access help by using the `help()` function or by pressing `F1` to open the Python documentation for the currently selected item.

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++?

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

In general, a wrapper is a function or class that provides additional functionality or modifies the behavior of an existing feature or object. The wrapped feature refers to the original feature or object being modified or extended. A decorator, on the other hand, is a specific type of wrapper that dynamically modifies or enhances the behavior of a function or class by wrapping it with additional functionality. Decorators are often used in Python using the `@decorator` syntax.

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

A generator function, when called, returns a generator object. This generator object can be iterated over to generate a sequence of values on-demand using the `yield` keyword within the function. Each time the generator's `next()` method is called, the function's execution resumes from where it left off and yields the next value in the sequence.

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?

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` statement with `yield` statements. By using the `yield` keyword, the function can generate a sequence of values and suspend its execution state, allowing it to be resumed later to produce the next value in the sequence.

Q12. Identify at least one benefit of generators.

One benefit of generators in Python is their ability to generate values on-demand, which saves memory and improves efficiency. Instead of generating and storing all values in advance, generators produce values dynamically, as they are needed. This is especially useful when dealing with large or infinite sequences, as it avoids the need to store the entire sequence in memory, resulting in improved performance and reduced memory usage.