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

at the runtime?

Q2. What is the smallest number of statements you&#39;d have to write in most programming languages to

replace the Python expression a, b = a + b, a?

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

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.

Q5. If you&#39;re using IDLE to run a Python application, explain how to print a multidimensional list as

efficiently?

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

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

possible from inside IDLE?

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&#39;t do in C or

C++?

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

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

Q12. Identify at least one benefit of generators.

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

The += operator is not just for show; it has practical use and can lead to more efficient results under certain conditions. The += operator performs in-place addition when possible, which can be more efficient than creating and reassigning new objects.

* **In-place Operations**: For mutable objects (like lists), += modifies the object in place rather than creating a new one.
* **Efficiency**: For some data types (e.g., lists), in-place operations can be faster because they avoid the overhead of creating new objects and copying data.

**Example**:

python

# Using += with a list (mutable object)

my\_list = [1, 2, 3]

my\_list += [4, 5] # Modifies the list in place

print(my\_list) # Outputs: [1, 2, 3, 4, 5]

### **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, replacing the Python expression a, b = a + b, a typically requires at least three statements:

1. **Temporary Variable**: To store the intermediate result.
2. **Update**: To assign the result to the first variable.
3. **Assignment**: To update the second variable.

**Example in C++**:

cpp

int temp = a + b;

b = a;

a = temp;

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

The most effective way to create and initialize a list of 100 integers all set to 0 is to use list multiplication.

**Example**:

python

my\_list = [0] \* 100 # Creates a list with 100 elements, all set to 0

### **Q4. What is the most effective way to initialize a list of 99 integers that repeats the sequence 1, 2, 3?**

You can use list multiplication combined with slicing to achieve this efficiently.

**Example**:

python

sequence = [1, 2, 3]

my\_list = sequence \* (99 // len(sequence)) + sequence[:(99 % len(sequence))]

**Explanation**:

1. **Multiply**: sequence \* (99 // len(sequence)) repeats the sequence enough times to cover most of the list.
2. **Add Remaining**: sequence[:(99 % len(sequence))] adds any remaining elements needed to reach 99 elements.

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

To print a multidimensional list in IDLE efficiently, you can use nested loops or list comprehensions to format the output.

**Example**:

python

matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

for row in matrix:

print(row)

This will print each row of the matrix on a new line.

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

Yes, you can use list comprehension with a string to create a list based on its characters or to transform the string.

**Example**:

python

s = "hello"

chars = [char.upper() for char in s] # Converts each character to uppercase

print(chars) # Outputs: ['H', 'E', 'L', 'L', 'O']

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

From the command line, you can get support for a user-written Python program using the --help option, provided that your script supports it.

**Example**:

bash

python my\_script.py --help

Inside IDLE, you can access help for a script by using the help() function if you have documented your script with docstrings.

**Example**:

python

def my\_function():

"""This is a help message for my\_function."""

pass

help(my\_function)

### **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++?**

In Python, functions are first-class objects, meaning they can be:

* **Assigned to Variables**: You can store functions in variables.
* **Passed as Arguments**: Functions can be passed as arguments to other functions.
* **Returned from Other Functions**: Functions can return other functions.
* **Stored in Data Structures**: Functions can be stored in lists, dictionaries, etc.

**Example**:

python

def outer\_function():

def inner\_function():

return "Inner"

return inner\_function

func = outer\_function() # func is now a reference to inner\_function

print(func()) # Outputs: Inner

In C++ or Java, functions are not first-class objects and cannot be assigned, passed around, or returned in this way.

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

* **Wrapper**: A wrapper is a function or object that "wraps" around another function or object to extend or modify its behavior.
* **Wrapped Feature**: The feature that is being extended or modified by the wrapper.
* **Decorator**: A specific type of wrapper in Python that is used to modify or extend the behavior of functions or methods. Decorators are applied using the @decorator\_name syntax.

**Example**:

python

def decorator\_function(original\_function):

def wrapper\_function():

print("Wrapper code")

original\_function()

return wrapper\_function

@decorator\_function

def say\_hello():

print("Hello")

say\_hello() # Outputs: Wrapper code \n Hello

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

A generator function returns a **generator object**. This object can be iterated over to produce values one at a time, using the yield statement within the function.

**Example**:

python

def my\_generator():

yield 1

yield 2

yield 3

gen = my\_generator()

print(next(gen)) # Outputs: 1

print(next(gen)) # Outputs: 2

### **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?**

To make a function a generator function, you must use the yield statement inside the function instead of return. The yield statement allows the function to produce a series of values lazily.

**Example**:

python

def my\_generator():

yield 1

yield 2

yield 3

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

**Benefit of Generators**:

* **Memory Efficiency**: Generators produce values one at a time and do not require storing the entire sequence in memory. This is especially useful for working with large datasets or streams of data.

**Example**:

python

def large\_range(n):

for i in range(n):

yield i

gen = large\_range(1000000) # Uses memory efficiently for a large range