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

ANS:

Python’s assignment operators allow you to define **assignment statements**. This type of statement lets you create, initialize, and update variables throughout your code. Variables are a fundamental cornerstone in every piece of code, and assignment statements give you complete control over variable creation and mutation.

he compound-assignment operators combine the simple-assignment operator with another binary operator. Compound-assignment operators perform the operation specified by the additional operator, then assign the result to the left operand. For example, a compound-assignment expression such as

expression1 += expression2

can be understood as

expression1 = expression1 + expression2

However, the compound-assignment expression is not equivalent to the expanded version because **the compound-assignment expression evaluates expression1 only once, while the expanded version evaluates expression1 twice**: in the addition operation and in the assignment operation.

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:

When dealing with expressions in python, everything to the right of the ‘=’ operator, i.e. the assignment operator, is evaluated first then assigned to the variables to the left.

Additionally, whenever you use a comma on the RHS, you’re telling python to create a tuple. Using a comma on the LHS tells python to unpack the tuple on the RHS into the variables on the LHS.

For example, if we wrote the following into the python interpreter:

1. >>> a = 1
2. >>> b = 2
3. >>> a, b = b, a + b

The right side would get evaluated first.

In line 1, the right side evaluates to 1. Then we assign 1 to a.

In line 2, the right side evaluates to 2. Then we assign 2 to b.

In line 3, the right side evaluates to a tuple (2, 3). We then unpack this tuple and assign the two values to a and b respectively.

This method of assignment is very useful for swapping variables. For example, in most programming languages, to swap two variables, we would have to do the following:

1. >>> a = 1
2. >>> b = 2
3. >>> temp = a
4. >>> a = b
5. >>> b = temp

However, in python you could instead do:

1. >>> a = 1
2. >>> b = 2
3. >>> a, b = b, a

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

ANS:

 list(range(100, 0,-1)) function call.

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:

Given an array of integers **arr[]**, The task is to find the index of first repeating element in it i.e. the element that occurs more than once and whose index of the first occurrence is the smallest.

**Input:**arr[] = {10, 5, 3, 4, 3, 5, 6}  
**Output:** 5   
**Explanation:**5 is the first element that repeats

**Input:**arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10}  
**Output:**6   
**Explanation:**6 is the first element that repeats

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

ANS:

There can be more than one additional dimension to [lists in Python](https://www.geeksforgeeks.org/python-list/). Keeping in mind that a list can hold other lists, that basic principle can be applied over and over. Multi-dimensional lists are the lists within lists. Usually, a [dictionary](https://www.geeksforgeeks.org/python-dictionary/) will be the better choice rather than a multi-dimensional list in Python.

Accessing a multidimensional list:

**Approach 1:**

|  |
| --- |
| # Python program to demonstrate printing  # of complete multidimensional list  a = [[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]  print(a) |

Output:

[[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]

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

ANS:

List comprehension in Python is an easy and compact syntax for creating a list from a string or another list. It is a very concise way to create a new list by performing an operation on each item in the existing list. List comprehension is considerably faster than processing a list using the for loop.

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

ANS:

Using the command line to make IDLE pop up with a filename.py to run; ie >>> idle filename.py. From the command line, just do “python filename.py”. If you want to edit the file with Idle (and then optionnally run it), you can do “python -m idlelib filename.py”. Windows, Linux, Mac OS or something else.

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:

C++ does not support automatic garbage collection of resources. Python, on the other hand, has a feature of automatic garbage collection. Its memory management is system-controlled. We cannot do rapid prototyping using C++.

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

ANS:

Function wrappers are useful tools for modifying the behavior of functions. In Python, they're called decorators. Decorators allow us to extend the behavior of a function or a class without changing the original implementation of the wrapped function.

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

ANS:

In Python, a generator is a function that returns an iterator that produces a sequence of values when iterated over. Generators are useful when we want to produce a large sequence of values, but we don't want to store all of them in memory at once.

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:

In Python, a generator is a [function](https://www.programiz.com/python-programming/function) that returns an [iterator](https://www.programiz.com/python-programming/iterator) that produces a sequence of values when iterated over.

Generators are useful when we want to produce a large sequence of values, but we don't want to store all of them in memory at once.

## Create Python Generator

In Python, similar to defining a [normal function](https://www.programiz.com/python-programming/function), we can define a generator function using the def keyword, but instead of the return statement we use the yield statement.

def generator name(arg):

# statements

yield something

Q12. Identify at least one benefit of generators.

ANS:

Advantages of using Generators

Memory is saved as the items are produced when required, unlike normal Python functions. This fact becomes very important when you need to create a huge number of iterators. This is also considered as the biggest advantage of generators. Can be used to produce an infinite number of items.