Python Advanced Assignment-14

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

Ans.

No, the `+=` operator is not just for show. It can lead to more efficient runtime performance in some cases. For mutable objects like lists, using `+=` modifies the object in place rather than creating a new object, which can save both memory and time. For immutable types like strings, `+=` creates a new object but still provides a convenient and concise syntax.

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.

In most programming languages that don’t support tuple unpacking, you'd need two separate statements to achieve the same result as `a, b = a + b, a`. For example:

```c
temp = a;
a = a + b;
b = temp;
```

This uses a temporary variable to store the old value of `a` and ensure that both `a` and `b` are updated correctly.

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

Ans.

The most efficient way is to use list multiplication:

```
my_list = [0] * 100
```

This creates a list of 100 integers, all set to 0, in one statement.

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

Ans.

You can use list multiplication and slicing to accomplish this:

sequence = [1, 2, 3] * 33 # Repeat the sequence 1, 2, 3 thirty-three times to make 99 elements

This will create a list of length 99 with the repeating sequence.

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

Ans.

To print a multidimensional list efficiently in IDLE, you can use a loop or the `pprint` (pretty print) module to format the output:

from pprint import pprint

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

pprint(matrix)

This ensures that the list is printed in a readable, formatted manner.

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

Ans.

Yes, you can use list comprehension with a string. For example, to create a list of the uppercase characters in a string:

string = "hello world"

uppercase_chars = [char.upper() for char in string]

print(uppercase_chars)

This will return a list of all the characters in uppercase.

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

Ans.

From the command line, you can use the `help()` function or the `pydoc` command to get documentation or help on a user-written Python program:

python -m pydoc my_module

In IDLE, you can access the same help by typing `help()` in the interactive shell and passing the function, class, or module name.

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, because functions are first-class objects, you can:

- Assign functions to variables.

- Pass functions as arguments to other functions.

- Return functions from other functions.

- Store functions in data structures like lists or dictionaries.

This is not directly supported in languages like C++ or Java.

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

Ans.

- Wrapper: A wrapper is a function or class that wraps another function or object, adding additional functionality or altering behavior without modifying the original object.

- Wrapped feature: The wrapped feature refers to the original function or object that is being wrapped and whose behavior is potentially being altered or extended.

- Decorator: A decorator is a higher-order function that takes a function (the wrapped feature) and returns a new function (the wrapper) with added behavior.

In Python, decorators are often implemented using the `@decorator` syntax to wrap a function with additional logic.

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

Ans.

A generator function returns a generator object. This object can be iterated over, producing values lazily (one at a time) using the `yield` statement within 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. To make a function a generator function, replace `return` with `yield`. The `yield` statement allows the function to pause and return an intermediate result, resuming where it left off when the generator is called again.

Example:

```
def my_generator():
    yield 1
    yield 2
    yield 3
```

Q12: Identify at least one benefit of generators.

Ans.

One benefit of generators is memory efficiency. Since generators produce items one at a time, they are especially useful when working with large data sets where loading all items into memory at once would be inefficient or impractical.