**FSDS MAY BATCH 2022(Python Assignment -14)**

**Submitted by: Shubham Tiwari**

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

Ans: An assignment operator like += is not only for show, it can in fact lead to faster results at runtime in Python. This is because using the += operator modifies the existing object in place, rather than creating a new object and reassigning the variable to that object. This can save time and memory, especially when working with large data structures. Additionally, some python implementations will optimize certain operations like += and make them even faster.

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: To replace the Python expression **a, b = a + b, a** in most programming languages, you would need at least two statements.

One statement would be used to calculate the sum of **a** and **b** and assign it to a temporary variable (e.g., **temp = a + b**). The second statement would then be used to reassign the values of **a** and **b** based on the temporary variable (e.g., **a = temp, b = a**)

However, some languages have a built-in swap function that can do this in one statement, in that case it would only require one statement. For example in C++ **std::swap(a,b)**.

It's worth noting that the specific implementation and language used may change the exact number of statements required.

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

Ans: In Python, there are a few ways to set a list of 100 integers to 0. The most effective way would be to use a list comprehension, as it is both concise and efficient:

**my\_list = [0 for \_ in range(100)]**

Another way to achieve this is by using the multiply function of the itertools module:

**from itertools import repeat**

**my\_list = list(repeat(0, 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.

Ans: One way to accomplish this is to use the **\*** operator to repeat a list containing the desired sequence. Here is an example:

**my\_list = [1, 2, 3] \* 33**

**print(my\_list)**

**[1, 2, 3, 1, 2, 3, 1, 2, 3, ..., 1, 2, 3]**

**len(my\_list)**

**99**

In this example, the **\*** operator is used to repeat the list **[1, 2, 3]** 33 times, resulting in a list of 99 elements that contains the desired sequence.

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

Ans: There are a few ways to print a multidimensional list in Python using IDLE, but one efficient method is to use nested for loops to iterate through each element of the list and print them. Here is an example:

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

**for sub\_list in my\_list:**

**for item in sub\_list:**

**print(item)**

**OUTPUT:**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

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

Ans: Yes, it is possible to use list comprehension with a string. One way to do this is to iterate over the characters in the string and add them to a new list. For example:

**string = "Hello"**

**list\_of\_chars = [char for char in string]**

**print(list\_of\_chars)**

**Output: ['H', 'e', 'l', 'l', 'o']**

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

Ans: To get support with a user-written Python program from the command line, we can use the **python -m pydoc** command followed by the name of the module or function you need help with. For example, **python -m pydoc os** will display documentation for the **os** module.

we can also use the **help()** function within the Python interpreter to access documentation for a specific module or function. For example, **import os** then **help(os)** will display the documentation for the **os** module.

It is also possible to get support from inside IDLE, the Python IDE. We can access documentation for a specific module or function by typing the name of the module or function followed by a "?" and pressing enter in the interactive shell. For example, **os?** will display documentation for the **os** module.

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, functions are first-class objects, which means that they can be:

* Assigned to a variable
* Passed as an argument to a function
* Returned as a value from a function
* Stored in a data structure such as a list or a dictionary

These capabilities are not available in languages like C++ or Java, where functions are not treated as first-class citizens.

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

Ans: **A wrapper, a wrapped feature, and a decorator are all related concepts in programming, but they have slightly different meanings and uses.**

**A wrapper is a function or class that "wraps" or encapsulates another function or class, often adding additional functionality or behavior.** A wrapped feature is the original function or class that is being wrapped. For example, a wrapper function might take a list as an argument, and return a new list with some additional elements. The original list would be the wrapped feature.

**A decorator is a specific type of wrapper function in Python. It's a function that modifies the behavior of another function, usually by adding some sort of functionality before or after the original function is called.** The decorator function is applied to the original function using the "@" symbol followed by the decorator function name.

**In summary, a wrapper is a general concept of wrapping a feature and adding functionality, while a decorator is a specific type of wrapper in Python which modifies the behavior of a function by adding functionality before or after the function is called.**

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

Ans: A generator function is a special type of function in Python that uses the yield keyword instead of return. When a generator function is called, it returns a generator object, which is an iterator that can be used to iterate over a sequence of values. Instead of returning a single value and terminating, a generator function can be paused and resumed multiple times, allowing it to generate a sequence of values over time. Each time the generator's next() method is called, the function runs until it encounters a yield statement, at which point it returns the value of the yield expression and "pauses" the function's execution. The next time the next() method is called, the function resumes execution immediately after the most recent yield statement.

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 order for a function to become a generator function in Python, the "yield" keyword must be used at least once within the function.

Q12. Identify at least one benefit of generators.

Ans: One benefit of generators is that they allow for memory-efficient iteration through large data sets, because they do not load all of the data into memory at once. Instead, they generate the next value in the sequence on-the-fly, which can save a lot of memory, especially when dealing with very large data sets.