**Python Advance Assignment-14**

**1. Is an assignment operator such as += only a convenience? Can it actually result in faster performance at run time?**

Ans.

**+=** adds a number/string to a variable, changing the variable itself in the process. Meaning, ( += ) adds the value of the right operand to a variable and assigns the result to the variable.

Not really it result in faster performance because it take lesser time to operate and we dont require a new

**2. In most computer languages, what is the minimum number of statements you’d need to write instead of the Python statement a, b = a + b, a?**

Ans. In C programming it takes minimum 3 number of statement.

c=a+b

a=c

b=a

**3. What’s the most efficient way to initialize a list of 100 integers to 0 in Python?**

Ans. By using list comprehension in reverse order

l**=**[i **for** i **in** range(100,**-**1,**-**1)]

print(l)

[100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

**4. What’s the most efficient way of initializing a list of 99 integers with the pattern 1, 2, 3 repeated? Show precisely how to do that, if possible.**

l**=**[i **for** i **in** range(1,4)]

l2**=**[]

**for** i **in** range(33):

l2**.**extend(l)

print(l2,'\n')

print(f'Number of integers ares: {len(l2)}')

[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

Number of integers ares: 99

**5. If you’re running a Python program from within IDLE, describe how to most efficiently print a multidimensional list.**

Ans.

It is very simple procedure. We first need a variable and assign it a multidimensional list.

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

print(a)

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

**6. Can you use list comprehension on a string? If so, how?**

l**=**[i **for** i **in** 'shivansh']

print(l)

['s', 'h', 'i', 'v', 'a', 'n', 's', 'h']

**7. How can you get help on a user-written Python program from the command line? From within IDLE?**

*# Ans. We will use help method in that case. Example is as folowed*

help(print)

Help on built-in function print in module builtins:

print(...)

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

**8. Functions are said to be “first-class objects” in Python but not in most other languages, such as C++ or Java. What is something you can do with a Python function (callable object) that you cannot do in C or C++?**

Ans.

In python there is a beautiful things which we can do with the function. It is called first class function because of the following reasons:

1) It can be use as an argument

2) It can be use in return by some other function

3) It can be act as variable and we can assign some value to it

**9. What’s the difference between a wrapper, a wrapped function, and a decorator?**

Ans.

**Wrappers**: It also refers to decorators. It allows programmers to modify the behavior of function or class.

**Wrapper Function**: Wrapper functions can be used as an interface to adapt to the existing codes, so as to save from modifying the codes back and forth. It helps in allowing some codes to run repeadtedly.

**Decorators**: It allows to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it.

**10. When a function is a generator function, what does it return, if anything?**

Ans.

Generator function is a kind of a iterable function which knows that what i have generated the last time and what it will give me the next time. Using return or print statement we can simply get every element which we have iterated upon immediately at once. But what if we want that we get the elemenet one by one as per our requrement. Here generator comes.

We can also say that Generator is something which help in creating iterator sort of thing. We use **yield** keyword in place of **return** when we are dealing with generators. With the help of generator only it is not necessary to create an itertor from an iterables. We can make any thing an itertaor object.

It do not stores all the items at once. Whatever the value we need, it catches and throw it out. Means whenever we need a next value to print we just call a next function over its object and it will give that value.

**11. From the standpoint of the Python language, what is the one change that needs to be made to a function to turn it into a generator function?**

Ans.

Just replace **return** with **yield**

**def** gen():

**for** i **in** range(10):

**yield** i

a**=**iter(gen())

a

<generator object gen at 0x000001E36938A6C8>

next(a)

0

next(a)

1

**for** i **in** a:

print(i)

2

3

4

5

6

7

8

9

**12. Name at least one advantage of generators.**

Ans

It helps in creating an iterator which do not store all the items at once. This saves the memory and also system do not get hang.

It takes the value one by one when we required and throw it out one by one by using **next** function when and only we required. It doesnot give the whole items at onces. It onlt gives when it is needed.