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

Ans:-The Python += Operator. The Python += operator **adds two values together and assigns the final value to a variable**. This operator is called the addition assignment operator.

Let's run some time tests\* with [timeit.timeit](http://docs.python.org/2/library/timeit.html" \l "timeit.timeit):

>>> from timeit import timeit

>>>

>>> # Test 1

>>> test = """

... my\_list = []

... for i in xrange(50):

... my\_list.append(0)

... """

>>> timeit(test)

22.384258893239178

>>>

>>> # Test 2

>>> test = """

... my\_list = []

... for i in xrange(50):

... my\_list += [0]

... """

>>> timeit(test)

34.494779364416445

>>>

>>> # Test 3

>>> test = "my\_list = [0 for i in xrange(50)]"

>>> timeit(test)

9.490926919482774

>>>

>>> # Test 4

>>> test = "my\_list = [0] \* 50"

>>> timeit(test)

1.5340533503559755

>>>

As you can see above, the last method is the fastest by far.

However, it should *only* be used with immutable items (such as integers). This is because it will create a list with references to the same item.

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:-

Two

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

Ans:- my\_list = [0] \* 100

Print(my\_list)

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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.

Ans:- l1 **=** [1,2,3] *# create a list containing 1,2,3*

count **=** int(101**/**len(l1)) *# repeat count*

l **=** l1 **\*** count *# \* operator is used to repeat the list*

print(l)

[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]

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

Ans:- l **=** [[1,2,3,4],['A','B'],[5,6,7,8]]

l

[[1, 2, 3, 4], ['A', 'B'], [5, 6, 7, 8]]

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

Ans:- s **=** 'anjali'

[i**\***2 **for** i **in** s]

['aa', 'nn', 'jj', 'aa', 'll',’ii’]

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

Ans:- Steps for command line:

1. Open command prompt.

2. Go to the directory containing python program file(.py)

3. Type "python file\_name.py", press enter to run the file.

Steps for python IDLE:

1. Open python IDLE.

2. Click on file>open>file\_name.py

3. Press F5 key to run the program.

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 like any other object, such as an int or a list. That means that you can use functions as arguments to other functions, store functions as dictionary values, or return a function from another function. This leads to many powerful ways to use functions.

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 same. They can be thought of as functions which modify the functionality of another function. They help to make your code shorter and more "Pythonic".

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

Ans:- If a function is a generator function, it does not return any values, instead, it return an iterator object, which we can iterate through to get values.

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:- We can use "yield" keyword instead of "return" to make function a generator.

In [8]:

**def** even\_nos(n):

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

**if** i **%** 2 **==** 0:

**yield** i

**for** i **in** even\_nos(10):

print(i)

0

2

4

6

8

Q12. Identify at least one benefit of generators.

Ans:- The main advantage here is that instead of having to compute an entire series of values up front, the generator computes one value and then suspends its activity awaiting the next instruction. This feature is known as state suspension.

Generators are memory efficient.