Assignment 22 Solutions

Q1. What are the benefits of the built-in array package, if any?

Ans: Arrays represent multiple data items of the same type using a single name. In arrays, the elements can be accessed randomly by using the index number. Arrays allocate memory in contiguous memory locations for all its elements. Hence, there is no chance of extra memory being allocated in case of arrays. This avoids memory overflow or shortage of memory in arrays.

Lists and NumPy Arrays usually are much better alternatives. That's also the reason why rarely people knows Python has a built-in Array type.

However, when something is restricted, it must bring some benefits. That is the so-call "trade-off". In this case, Python Array will have a smaller size than Python List. This is demonstrated below.

In [2]:

```
from array import array
list_large = list(range(0, 100000))
arr_large = array('I', list_large)

arr_large.__sizeof__()  #Size comes out to be 400064
list_large.__sizeof__() #Size comes out to be 800040
```

Out[2]:

```
800040
```

In [3]:

```
#Using Built in Array
import array as built_in
array1 = built_in.array('i', [1, 2])
array1.append(3)
array1.append(4)
print(array1)
```

```
array('i', [1, 2, 3, 4])
```

Q2. What are some of the array package's limitations?

Ans: An array which is formed will be homogeneous. While declaring an array, passing size of an array is compulsory, and the size must be a constant.

Shifting is required for insertion or deletion of elements in an array.allocating more memory than the requirement leads to wastage of memory space and less allocation of memory also leads to a problem

Q3. Describe the main differences between the array and numpy packages ?

Ans: The array package doesn't provide any help with numerical calculation with the items insdie it in number form while NumPy give you a wide variety of numerical operations.

An array is a single dimensional entity which hold the numerical data, while numpy can have more than 1 dimension.

In case of array, item can be accessed by its index position and it is easy task while in numpy item is accessed by its column and row index, which makes it slightly time taking. Same goes with appending operation.

In case of array we do not form a tabular structure, while in numpy it forms a tabular structure

Q4. Explain the distinctions between the empty, ones, and zeros functions ?

Ans: The distinctions between the `empty` , `ones` , and `zero` functions are as follows :

- `Empty function:` An empty function is a function that does not contain any statement within its body. If you try to write a function definition without any statement in python ,it will return an error. To avoid this, we use pass statement. pass is a special statement in Python that does nothing. It only works as a dummy statement.
- `Ones:` This function returns a new array of given shape and data type, where the element's value is 1.
- `Zeros:` This function returns a new array of given shape and data type, where the element's value is 0.

Q5. In the fromfunction function, which is used to construct new arrays, what is the role of the callable argument?

Ans: fromfunction() function construct an array by executing a function over each coordinate and the resulting array, therefore, has a value fn(x, y, z) at coordinate (x, y, z). Parameters : function : [callable] The function is called with N parameters, where N is the rank of shape.

Q6. What happens when a numpy array is combined with a single-value operand (a scalar, such as an int or a floating-point value) through addition, as in the expression `A + n` ?

Ans: If any scaler value such as integer is added to the numpy array then all the elements inside the array will add that value in it.

In [5]:

```python
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr + 1)
```

```
[2 3 4 5 6]
```

Q7. Can array-to-scalar operations use combined operation-assign operators (such as += or *=)? What is the outcome ?

Ans: NO - It will carry out provided operation on all elements of array.

In [7]:

```python
#using scalar operations error came
import numpy as np

arr = np.array((1, 2, 3, 4, 5))
a = arr += 1
print(a)
```

```
  File "C:\Users\abhik\AppData\Local\Temp/ipykernel_20156/4216562642.py", line 5
    a = arr += 1
            ^
SyntaxError: invalid syntax
```

Q8. Does a numpy array contain fixed-length strings? What happens if you allocate a longer string to one of these arrays ?

Ans: : Yes, it is possible that we can include a string of fixed length in numpy array. The dtype of any numpy array containing string values is the maximum length of any string present in the array.Once set, it will only be able to store new string having length not more than the maximum length at the time of the creation. If we try to reassign some another string value having length greater than the maximum length of the existing elements, it simply discards all the values beyond the maximum length accept upto those values which are under the limit.

Q9. What happens when you combine two numpy arrays using an operation like addition (+) or multiplication (*)? What are the conditions for combining two numpy arrays ?

Ans: It will simply add or multiply element to element at same position.

NumPy's concatenate function can be used to concatenate two arrays either row-wise or column-wise. Concatenate function can take two or more arrays of the same shape and by default it concatenates row-wise i.e. axis=0. The resulting array after row-wise concatenation is of the shape 6 x 3, i.e. 6 rows and 3 columns.

In [9]:

```python
import numpy as np

arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = arr1+arr2
arr1 = arr1*arr2

print(arr)
print("\n",arr1)
```

```
[[ 6  8]
 [10 12]]

 [[ 5 12]
 [21 32]]
```

Q10. What is the best way to use a Boolean array to mask another array?

Ans: Boolean masking is typically the most efficient way to quantify a sub-collection in a collection. Masking in python and data science is when you want manipulated data in a collection based on some criteria. The criteria you use is typically of a true or false nature, hence the boolean part.

Using `masked_where()` function: Pass the two array in the function as a parameter then use `numpy. ma. masked_where()` function in which pass the condition for masking and array to be masked.

Using `masked_where()` , `getmask()` and `masked_array()` function: Pass the two array in the function as a parameter then use numpy. ma.

Q11. What are three different ways to get the standard deviation of a wide collection of data using both standard Python and its packages? Sort the three of them by how quickly they execute ?

Ans: `std()` from Numpy package, `stdev()` from Statistics package and the third way is to write our own code as done with the help of math package below:

In [11]:

```python
import numpy as np #for declaring an array

def mean(data):
    n = len(data)
    mean = sum(data) / n
    return mean

def variance(data):
    n = len(data)
    mean = sum(data) / n
    deviations = [(x - mean) ** 2 for x in data]
    variance = sum(deviations) / n
    return variance

def stdev(data):
    import math
    var = variance(data)
    std_dev = math.sqrt(var)
    return std_dev

data = np.array([4, 8, 6, 5, 3, 2, 8, 9, 2, 5, 12])

print("Standard Deviation of the sample is % s "% (stdev(data)))
```

```
Standard Deviation of the sample is 3.009626428590336
```

12. What is the dimensionality of a Boolean mask-generated array ?

Ans: It will have same dimensionality as input array. Masking comes up when we want to extract, modify, count, or otherwise manipulate values in an array based on some criterion: for example, we might wish to count all values greater than a certain value, or perhaps remove all outliers that are above some threshold. In NumPy, Boolean masking is often the most efficient way to accomplish these types of tasks.

In [12]:

```python
import numpy as np

arr = np.asarray([[[[1, 11], [2, 22], [3, 33]],
                   [[4, 44], [5, 55], [6, 66]]],
                  [[[7, 77], [8, 88], [9, 99]],
                   [[0, 32], [1, 33], [2, 34]]]])

masked_arr = np.ma.masked_less(arr, 3)

print(masked_arr)
```

```
[[[[-- 11]
   [-- 22]
   [3 33]]

  [[4 44]
   [5 55]
   [6 66]]]


 [[[7 77]
   [8 88]
   [9 99]]

  [[-- 32]
   [-- 33]
   [-- 34]]]]
```