In [1]:

```
import numpy as np
names = np. array(['Bob' , 'Joe' , 'Boby','Will' ,'Willy' , 'Joe' , 'Joe' ])
data=np.arange(3,10,1)
print(data[names=="Boby"])
```

```
[5]
```

In [ ]:

```
The data generation functions in numpy.random use a global random seed. To avoid global state,
you can use numpy.random.RandomState to create a random number generator isolated from others:
```

In [2]:

```
import numpy as np
arr = np. arange(6)
arr. reshape((2, -1))

#For Array reshaping one of the passed shape dimensions can be -1, in which case the value
used for that dimension will be inferred from the data. So in this question it will create
an array of size 2x3 to arrange all elements of the array.
```

Out[2]:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [ ]:

```
 a) Error
 b) Will create an array of size 3x2
 c) Will create an array of size 2x1
 d) Will create an array of size 2x3
```

In [ ]:

```
The opposite operation of reshape from one-dimensional to a higher dimension is typically
known as flttening or raveling:
```

In [ ]:

```
NumPy's library of algorithms written in the C language can operate on this memory without
any type checking or other overhead.
```

In [3]:

```
import numpy as np
arr = np. arange(10)
print(arr[5:8])
```

```
[5 6 7]
```

In [4]:

```
# What will be the output of the following code :
arr2d = np. array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d[1,1:])
```

```
[5 6]
```

In [5]:

```
# What will be the output of the following code :
import numpy as np
arr = np. arange(10)
arr_slice = arr[5: 8]
arr_slice[:] = 64
print(arr[3:7])
```

```
[ 3  4 64 64]
```

In [8]:

```
# What will be the output of the following code :
import numpy as np
arr = np.arange(4)
arr.reshape(2,2)
print(arr.ndim)
```

```
1
```

In [9]:

```
# What will be the output of the following code :
import numpy as np
arr = np. arange(10)
arr[5]=50
print(arr[4:7])
```

```
[ 4 50  6]
```

In [10]:

```
# What will be the output of the following code :
import numpy as np
arr = np.array([False, True, True, False])
arr.any()
```

Out[10]:

```
True
```

In [ ]:

```
9. Which Numpy Array creation function produce an array of the given shape and dtype with all values
set to the indicated “fill value”

Numpy.Full

```

In [ ]:

```
Which functionality is not present in Numpy

Some functionalities like Time Series manipulation is not present in NumPy. we have to use Pandas for that.
```

In [11]:

```
# What will be the output of the following code :
import numpy as np
arr=[-3.2623]
print(np.sqrt(arr))

```

```
[nan]

C:\Users\Nadeem\AppData\Local\Temp\ipykernel_20632\4231124872.py:4: RuntimeWarning: invalid value encountered i
n sqrt
  print(np.sqrt(arr))
```

In [12]:

```
# What will be the output of the following code :
import numpy as np
x = np. array([1. , 2. , 3. ])
y = np. array([[6.], [- 1], [8]])
print(" x. dot(y) = \n",x. dot(y))
```

```
 x. dot(y) = 
 [28.]
```

In [13]:

```
# What will be the output of the following code :
import numpy as np
arr = np.array([11, 2.5, 3.6,-87])
cond = np.array([True, False, True])
result = np.max(np.where(np.abs(arr) > 3, arr,0))
print(result)
```

```
11.0
```

In [ ]:

```
In Python random.seed() function is used to save the state of a random function, so that it
can generate _________________ on multiple executions of the code
 d) same random numbers

# Seed function is used to save the state of a random function, so that it can generate same
random numbers on multiple executions of the code.
```

In [ ]:

```
Which Numpy function will create a square N x N identity matrix

eye and identity Numpy functions creates a square N × N identity matrix (1s on the diagonal and 0s elsewhere
```

In [ ]:

```
In Numpy save and load functions are used for efficiently saving and loading array data on disk.
Arrays are saved by default in file with extension ___________

.npy
```

In [32]:

```
import numpy as np
arr1 = np.array([0, 1, 2, 3])
arr2 = [0, 2, 5]
res = np.in1d(arr1, arr2, invert = True)
print(res)
```

```
[False  True False  True]
```

Out[32]:

```
[0, 2, 5]
```

In [15]:

```
import numpy as np
a = np.array([1, 2, 3,4,5], ndmin = 2)
print(a.shape,a.ndim)
```

```
(1, 5) 2
```

In [16]:

```
import numpy as np
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.0])
print(arr.astype(np.int32))
```

```
[ 3 -1 -2  0 12 10]
```

In [17]:

```
import numpy as np
a = np.array([1, 2, 3,4,5])
print(a.shape,a.ndim)
```

```
(5,) 1
```

In [18]:

```
import numpy as np
arr = np.array([1, 2, 3,4])
print(arr.cumprod(axis=0))
```

```
[ 1  2  6 24]
```

In [19]:

```
Which statement best describes Numpy's ndarray ?

 c) ndarray, an efficient multidimensional array providing fast array-oriented arith- metic operations
```

```
  Input In [19]
    Which statement best describes Numpy's ndarray ?
          ^
SyntaxError: invalid syntax
```

In [20]:

```
We can explicitly convert or cast an array from one dtype to another using ndarray’s ________ method:
```

```
  Input In [20]
    We can explicitly convert or cast an array from one dtype to another using ndarray’s ________ method:
       ^
SyntaxError: invalid syntax
```

In [22]:

```
import numpy as np
a = np.array([1,2,3])
a
```

Out[22]:

```
array([1, 2, 3])
```

In [23]:

```
It’s not safe to assume that _______ will return an array of all
zeros.

It’s not safe to assume that np.empty will return an array of all zeros. In some cases,
it may return uninitialized “garbage” values

```

```
  Input In [23]
    It’s not safe to assume that _______ will return an array of all
      ^
SyntaxError: invalid character '’' (U+2019)
```

In [24]:

```
import numpy as np
x = [3,45,76,7,34]
y = [1,82,1,88,22]
z=np.maximum(x,y)
print(z)
```

```
[ 3 82 76 88 34]
```

In [ ]:

```
The most important object defined in NumPy is an N-dimensional array type called?

ndarray
```

In [25]:

```
import numpy as np
arr = np.arange(3, 22, 4)
print(arr)
```

```
[ 3  7 11 15 19]
```

In [26]:

```
arr2d = np. array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d[1,1:])
```

```
[5 6]
```

In [27]:

```
arr = np. arange(10)
arr_slice = arr[5: 8].copy()
arr_slice[:] = 64
print(arr[4:7])
```

```
[4 5 6]
```

In [28]:

```
arr1 = np.arange(4).reshape(2,2)
arr2 = np. array([[5,6], [7,8]])
res=np. hstack((arr1, arr2))
print(res)
# There are some convenience functions, like vstack and hstack, for common kinds of concatenation.
vstack can be called as vertical stack  whereas hstack can be called as horizontal stack. So the
answer of this question will be [[0 1 5 6] [2 3 7 8]]
```

```
[[0 1 5 6]
 [2 3 7 8]]
```

In [ ]:

```

```

In [33]:

```
data2 = [[1, 2, 3, 4,5,6], [5, 6, 7, 8,9,10]]
arr2 = np.array(data2).reshape(2,2,3)
arr2.ndim
```

Out[33]:

```
3
```

In [34]:

```
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
print(numeric_strings.astype(float))
```

```
[ 1.25 -9.6  42.  ]
```

In [ ]:

```
NumPy is often used along with packages like

d) Matplotlib
```

In [37]:

```
import numpy as np
arr = np.array([11, 2, 5,34])
print(arr.cumsum())
```

```
[11 13 18 52]
```

In [39]:

```
import numpy as np
arr = np.array([11, 2, 5,34])
print(arr.cumsum(axis=0))
```

```
[11 13 18 52]
```

In [38]:

```
import numpy as np
arr = np.array([11, 2, 5,34])
print(arr.cumsum(axis = 1))
# In multidimensional arrays, accumulation functions like cumsum return an array of the same size,
but with the partial aggregates computed along the indicated axis according to each lower dimensional slice.
In our axis is 1 which means on vertical axis but array is a single dimensional array so this code will gene
error "axis 1 is out of bounds for array of dimension "
```

```
---------------------------------------------------------------------------
AxisError                                 Traceback (most recent call last)
Input In [38], in <cell line: 3>()
      1 import numpy as np
      2 arr = np.array([11, 2, 5,34])
----> 3 print(arr.cumsum(axis = 1))

AxisError: axis 1 is out of bounds for array of dimension 1
```

In [40]:

```
import numpy as np
arr = np.arange(9).reshape((3,3))
print(arr)
print(arr[[0, 1], [0, 2]])
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[0 5]
```

In [41]:

```
import numpy as np
arr = np.arange(3)
arr=arr.repeat(3)
print(arr)
# Two useful tools for repeating or replicating arrays to produce larger arrays are the repeat and
tile functions. repeat replicates each element in an array some number of times, producing a larger array.
```

```
[0 0 0 1 1 1 2 2 2]
```

In [42]:

```
import numpy as np
a = np.array([1,2,3])
print (a)
```

```
[1 2 3]
```

In [43]:

```
import numpy as np
arr = np.array([1, 2, 3,4])
print(arr.cumprod(axis=0))
```

```
[ 1  2  6 24]
```

In [45]:

```
import numpy as np
arr = np. arange(9)
arr. reshape((2, 4))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [45], in <cell line: 3>()
      1 import numpy as np
      2 arr = np. arange(9)
----> 3 arr. reshape((2, 4))

ValueError: cannot reshape array of size 9 into shape (2,4)
```

In [46]:

```
import numpy as np
arr = np. arange(10)
print(arr[5:8])
```

```
[5 6 7]
```

In [47]:

```
arr = np.arange(8).reshape((2,4))
arr=arr.T
print(arr[0,1])
```

```
4
```

In [48]:

```
import numpy as np
arr=[-3.2623]
print(np.sign(arr))
# numpy.sign() function is used to compute the sign of each element. 1 (positive), 0 (zero), or -1 (negative
```

```
[-1.]
```

In [49]:

```
import numpy as np
data = np.random.randn(2,3)
print(data)
```

```
[[ 0.25685888  2.11079343 -0.04822527]
 [-0.16581439 -1.01122636 -0.90031617]]
```

In [ ]:

```
NumPy by itself does not provide

# NumPy by itself does not provide modeling or scientific functionality, having
an understanding of NumPy arrays and array-oriented computing will help you use
tools with array-oriented semantics, like pandas, much more effectively.
```

In [53]:

```
import numpy as np
x = [3,45,76,7,34]
y = [1,82,1,88,22]
z=np.maximum(x,y)
print(z)

# In this question universal function numpy.maximum() is used. numpy.maximum computes the element-wise
maximum of the elements in x and y. So same index element numbers are compared and which ever is higher
is placed in the result. In the same way all elements of the array are computerd
```

```
[ 3 82 76 88 34]
```

In [ ]:

In [ ]: