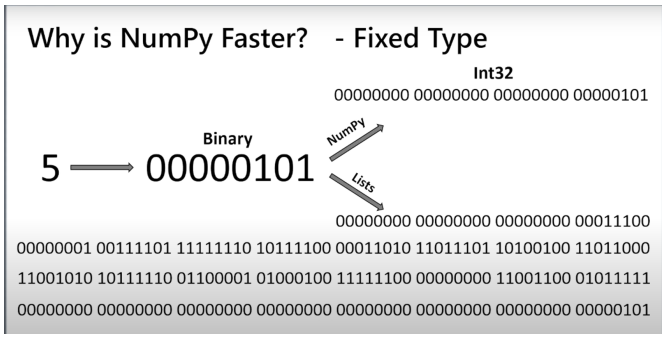
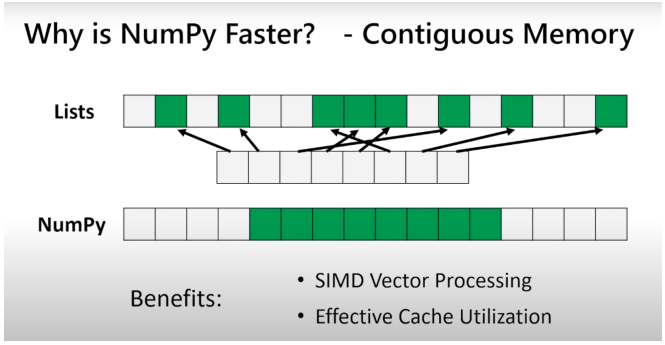
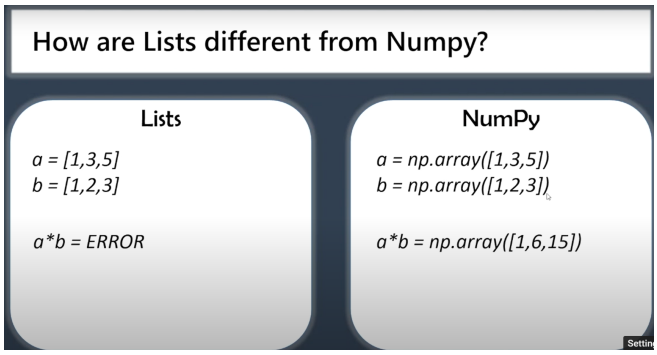
Numpy

- Multidimensional array library. (store data in 1D, 2D, 3D, … arrays)
- Why use Numpy over Lists? -> Numpy is fast, Lists are slow.
  -> Faster to read, less bytes of memory.
  -> No type checking when iterating through objects.
  -> Numpy is fixed type.
  -> Numpy has contiguous memory: SIMD (Single Instruction Multiple Data) Vector processing, Effective Cache Utilization
  Lists parameters: Size, Reference count, Object type, Object value







- Applications: 1. Mathematics (Matlab replacement)
  2. Scipy library contains even more mathematical tools.
  3. Plotting (Matplotlib)
  4. Backend (Pandas, Connect 4, Digital Photography)
  5. Machine Learning

Coding:
Initialize arrays: a = numpy.array([[1, 2, 3], [4, 5, 6]])
Initialize with specific size: b = np.array([1, 2, 3], dtype = 'int16') #it will take 2 Bytes for storage.
Get dimension: a.ndim
Get shape: a.shape
Get type: a.dtype #'int32' means 4 Bytes of storage
Get size: a.itemsize #int output in bytes
Get total size of array: a.nbytes

Accessing/ changing specific elements/ rows/ columns.
A[row_index, col_index] #row_index and col_index can be negative also, but both starts from 0
Specific row: A[row_index, :]
Specific column: A[: , col_index]
Fancy: [start_index : end_index + 1 : step_size]
Change element: A[row_index, col_index] = n
Change row/ col: A[row_index, :] = [n1, n2, ..]

Defining 3-D array:
Arr = np.array([[[n1, n2, n3], [n4, n5, n6]], [[n7, n8, n9], [n10, n11, n12]]]) #shape = (2, 2, 3)

Initializing different types of arrays:

1. Zeros matrix: np.zeros((n_rows, n_col)) #np.zeros(shape)
2. Ones matrix: np.ones((n_rows, n_col), dtype = 'int32') #it can be of any dimension
3. Any other number: np.full(shape, value) #np.full((2, 3), 99, dtype = 'float32')
4. Full_like: np.full(arr.shape, value) or, np.full_like(arr, value)
5. Random decimal no.s : np.random.rand(row_index, col_index) #eg. np.random.rand(4, 2) or, np.random.random_sample(arr.shape) #eg. no.random.random_sample((4, 2))
6. Random integer values: np.random.randint(4, size = (row_index, col_index)) #this will give you matrix with given size of integers within range 0, 3
   Or, np.random.randint(-4, 6, size = (2, 3)) #this will give values from -4 to 5, with arr size = a(2, 3)
7. Identity matrix: np.identity(n_rows/ col)
8. Repeat an array: np.repeat(arr, n_times, axis = 0/ 1)

XXX Caution when copying arrays XXX
If copied arrays changes, original also changes:

a = np.array([2, 3, 4])
b = a
b[1] = 1 #both a and b are pointing to same location, thus both's 1st index get's modified

Therefore, to avoid this:
b = a.copy()

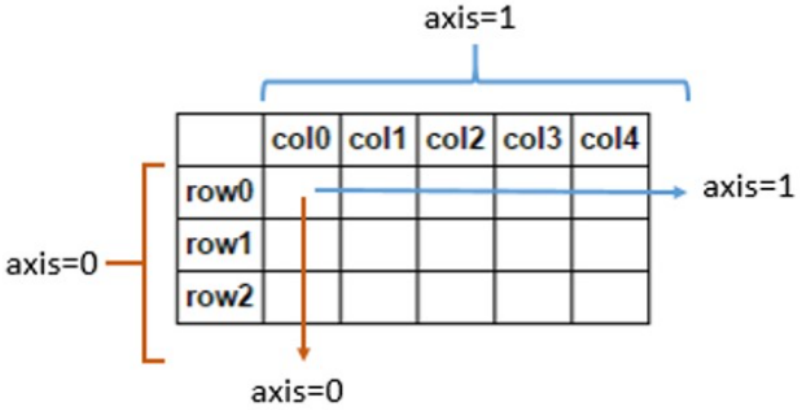
Mathematics:
a = np.array([0, 3, 4, 5, 5])
a + 2: adds each element with 2
a - 2: subracts each element with 2
a * 2: multiples each element with 2
a / 2: divides each element with 2
a ** 2: squares each element
b = np.array([2, 3, 4, 4, 2])
a + b: adds each corresponding array elements
np.sin(a): takes sine of each value of arr into another arr

Linear Algebra:

1. Matrix Multiplication: np.matmul(a, b) #here if shape(a) = (a1, a2) and shape(b) = (b1, b2),
   then a2 = b1 and shape(matmul) = (a1, b2).
2. Matrix Determinant: np.linalg.det(a) #a should be a square matrix
3. Singular Value Decomposition: np.linalg.svd(a)
4. Eigen values: np.linalg.eigvals(a)
5. Eigen values and Eigen vectors: np.linalg.eig(a)

Statistics:
np.min(a, axis = 0/ 1)
np.max(a, axis = 0/ 1)
np.sum(a, axis = 0/ 1)

Reorganizing arrays:



Reshape: a.reshape(new_shape) #The new shape should be compatible with the total no. of elements.

Vertical stacking: np.vstack([a1, a2]) #for vstack, n_cols should be same.
Horizontal stacking: np.hstack([b1, b2])

Miscellaneous:

Load data from file: filedata = np.genfromtxt('data.txt', delimiter = ' , ') #delimiter is separator
Convert file from float type to int: filedata = filedata.astype('int32')
Boolean masking and Advanced Indexing:

```
# You can index with a list in NumPy
a = np.array([1, 2, 3, 45, 6, 7, 8, 8, 5, 20])
a[[4, 5, 6]]

o/p: array([6, 7, 8])

#checking for elements:
```

1. filedata > 50

```
o/p: array([[False, False, False, False, False, False, False, False, False],
      [False, False, False, False, False, False, False, False, False],
      [False, False, False, False, False,  True, False,  True, False],
      [False, False, False, False, False, False, False, False,  True],
      [False, False, False, False, False, False,  True, False,  True]])
```

2. filedata[filedata > 50]

```
o/p: array([ 66,  66, 356,  67,  78])
```

3. np.any(filedata > 50, axis = 0)

```
o/p: array([False, False, False, False, False,  True,  True,  True,  True])
```

4. np.all(filedata > 2, axis = 1)

```
o/p: array([ True, False, False, False,  True])
```

5. ((filedata > 50) & (filedata < 100))

```
o/p: array([[False, False, False, False, False, False, False, False, False],
            [False, False, False, False, False, False, False, False, False],
            [False, False, False, False, False,  True, False,  True, False],
            [False, False, False, False, False, False, False, False, False],
            [False, False, False, False, False, False,  True, False,  True]])
```

6. ~((filedata > 50) & (filedata < 100))

```
o/p: array([[ True,  True,  True,  True,  True,  True,  True,  True,  True],
```

[True, True, True, True, True, True, True, True, True],
[True, True, True, True, True, False, True, False, True],
[True, True, True, True, True, True, True, True, True],
[True, True, True, True, True, True, False, True, False]])

Revision Questions: