NumPy

- Stands for Numeric Python
- Used for Scientific Computing, Data Analysis, Machine Learning and Image Processing
- We can think 2-D list like a matrix. For 3-D, think like a cube
- Main component is **n-d array of same data type**
- Faster than Python List
- Provides operation for Fourier Transform, Matrix & Linear Algebra
- Arithmetic operation are element wise

Create Array

- Can be created from existing list or NumPy function
- Example:
  - list_a = [1, 2, 5, 10] # check for mixed data type
  - arr_1 = np.array(list_a) **or** arr_1 = np.asarray(list_a, dtype=np.float64)
  - print(arr_1.shape) # Prints shape of array
  - print(arr_1.ndim) # Prints dimension of array
  - print(arr_1.dtype) # Prints data type of array element
  - print(np.floor(arr_1)) # Converts each element to floor value
  - print(np.ceil(arr_1)) # Converts each element to ceiling value
  - print(np.round(arr_1, 2)) # Rounds each element of array
  - len, join are valid

Indexing

- Accessing element in NumPy array is same as that of list
- In higher order array we can use , separated notation as well
- Example:
  - `single_array = np.array((4, 12, 7, 8, 10))`
  - `print(single_array[2])`

  - `nd_array = np.asarray([[1, 2, 3], [2, 3, 4], [3, 3, 1]])`
  - `print(nd_array[0][1])` # 2nd element of 1st array
  - `print(nd_array[0, 1])` # 2nd element of 1st array

Slicing

- Slicing NumPy array is same as that of list
- Example:

```
arr_1d = np.array([1, 4, 6, 7, 8, 10])
print(arr_1d[1:4])
print(arr_1d[1:4:2])
print(arr_1d[:3])
print(arr_1d[2:])
print(arr_1d[::-1])

nd_array = np.array([[1, 2, 3], [2, 3, 4], [3, 3, 1]])
print(nd_array[0][:2])
print(nd_array[0, :2])
```

Inserting

- For adding element in existing NumPy array
- Example:
  - `np_arr = np.array([3, 7, 8, 1, 4])`
  - `print(np_arr)`
  - `np_arr2 = np.insert(np_arr, 1, 5)`
  - `print(np_arr)`
  - `print(np_arr2)`

  - `np_2d_arr = np.array([[1, 2], [5, 2]])`
  - `np_2d_arr2 = np.insert(np_2d_arr, 1, [[5, 3], [3, 4]], axis=0)`
  - `print(np_2d_arr2)`

  - `np_2d_arr = np.array([[1, 2, 3], [5, 2, 1]])`
  - `np.insert(np_2d_arr, 1, [4, 7, 4]) # Try this with axis values`

Append

- Add new element at the end
- Example:
  - `np_arr = np.array([[1, 2], [5,2]])`
  - `np_arr2 = np.append(np_arr, [6, 1, 2])`
  - `print(np_arr2)`

  - `np_arr = np.array([[1, 2], [5,2]])`
  - `np_arr2 = np.append(np_arr, [[5, 3], [3, 4]], axis=1)`
  - `# Try with axis=0 and without axis`

Delete

- Delete data from NumPy array
- Example:
  - `np_arr = np.array([[1, 2], [5,2]])`
  - `np.delete(np_arr, 1)`
  - `np.delete(np_arr, 1, axis=0)`
  - `np.delete(np_arr, 1, axis=1)`

  - `np_arr = np.array([[[1, 2], [5,2]], [[3, 4], [7,8]]])`
  - `np.delete(np_arr, 1)`
  - `np.delete(np_arr, 1, axis=0)`
  - `np.delete(np_arr, 1, axis=1)`
  - `np.delete(np_arr, 1, axis=2)`

Numeric Operator

- Operation are performed element wise in NumPy array
- Supports operators like: + , - , /, //, **, *, %, @, <, >, ==
- @ ➔ Dot product or we can use np.dot(a_1, a2)
- Example:
  - arr_1 = np.array([[1, 2], [3, 4]])
  - arr_2 = np.array([[3, 2], [5, 0]])
  - arr_1 + arr_2
  - arr_1 ** 2 # Broadcasting

Broadcasting

- Smaller array is broadcasted over the larger one provided trailing dimension matches (or 1)
- Example:
  - `x = np.arange(0, 4 * np.pi, 0.1)`
  - `y = np.sin(x)  # or np.exp`

  - `arr_1 = np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3]])`
  - `arr_2 = np.array([0, 1, 2]) # np.array([[0], [1], [2]])`
  - `print(arr_1 + arr_2)`

Constants & Functions

- NumPy provides constants and functions for common uses
- Some NumPy constants: np.inf, np.nan, np.pi
- Some NumPy functions:
  - np.floor, np.ceil, round, sort, np.abs, np.square, np.power

Aggregate

- np.operation(array) or array.operation()
- sum, mean, min, max, var, std, cumsum, cumprod, argmax, argmin, clip
- median , cov(x,y), quantile(q), ptp, average(a, weights=w)
- Example:
  - `np_arr = np.array([[1, 4, 2], [2, 6, 1]])`
  - `np.sum(np_arr) # Try with axis option`

Filtering

- Used when we want to select element based on condition
- Result will be 1-D array
- We can combine condition with &, | and ~
- Example:
  - np_arr = np.array([[1, 2, 3], [2, 3, 5], [1, 3, 4], [2, 5, 1]])
  - np_filtered = np_arr[(np_arr > 2) | (np_arr < -2)]
  - np_arr[np_arr > 0].mean()
  - np_arr[np_arr < 0] = 0

where

- Unlike filter where returns indices where the condition is True. (Filter gives data)
- Usefult on simulating if else behavior
- np.where(mat_a < q1, q1, mat_a)

Select

- Categorize data based on multiple condition at once
- Doesn't require loop

```
import numpy as np

arr = np.array([5, 10, 12, 19, 20, 24, 30])

conditions = [arr < 13, (arr >= 13) & (arr < 20), arr >= 20]
choices = ["Child", "Teen", "Adult"]

categories = np.select(conditions, choices, default="Unknown")
print(categories)
```

Exercise

Generate data

- np_zero_array = np.zeros((2, 5))
- one_array = np.ones((3,6), dtype=int)
- full_array = np.full((3,6), 5, dtype=int)
- lin_array = np.linspace(1, 10, 4)
- log_array = np.logspace(1, 10, 4)
- geom_array = np.geomspace(1, 10, 4)
- range_array = np.arange(3, 9)
- random_array = np.random.random((5,5))
- random_array = np.random.randint(0, 10, size=(10, 10))

Generate data

- np.random.standard_normal((3,4))
- normal_dist_array = np.random.normal(mu, sigma, size=200)
- identity_array = np.eye(3, dtype=int)
- identity_matrix = np.eye(2, 3, dtype=int)
- identity_array = np.eye(3, k=-1, dtype=int)
- identity_array = np.identity(3, dtype=int)

Re-shape

- For changing shape of array
- Usually (when possible) we convert array into 1D, do all the operations and convert back to its original shape
- Example:
  - `single_array = np.arange(10)`
  - `re_shaped_array = np.reshape(single_array, (5,2))`
  - `multi_arr = np.reshape(single_array, (5,2), order='F')`

  - `multi_arr.reshape(10) # vs (1,10)`

Re-size

- Same as re-shape but if not compatible fills remaining position from initial
- If we resize to small shape excess elements will be left out
- Example:
  - single_array = np.arange(10)
  - resized = np.resize(single_array, (3,3))
  - sized_up = np.resize(resized, (4,4))
  - sized_up = np.zeros((4,4), dtype=resized.dtype)
  - sized_up[:3, :3] = resized

Flatten

- Returns the copy of array collapsed into 1D
- We can use flatten / reshape / ravel (returns view)
- Example:
  - arr = np.array([[1, 2], [3, 4]])
  - print(arr)
  - fltn = arr.flatten() # Order = 'F' optionally
  - fltn[0] = 5
  - print(fltn)
  - print(arr) # Try with Ravel it modifies element

Transpose

- We can use .T or np.transpose()
- Example:
  - arr = np.array([[1,2,3], [4,5,6]])
  - arr.T
  - np.transpose(arr)

Stack Array

- Add array on top of another array
- Can be done vertically or horizontally
- Example
  - arr_1 = np.array([[1, 2, 3], [4, 5, 6]])
  - arr_2 = np.array([[7, 8, 9], [10, 11, 12]])
  - np.hstack((arr_1, arr_2))
  - np.vstack((arr_1, arr_2))

Concatenation

- Combine two array
- Example:
  - arr_1 = np.array([[1, 2, 3], [4, 5, 6]])
  - arr_2 = np.array([[7, 8, 9], [10, 11, 12]])
  - np.concatenate((arr_1, arr_2))
  - np.concatenate((arr_1, arr_2), axis=1)

Copy Array

- Slicing array and making change on it affects original array
- We need to copy array and then later perform operation
- Example:
  - np_ar1 = np.array([[4,0, 32], [3,12, 59]])
  - sub_arr = np_ar1[:,1:]
  - sub_arr[0, 0] = 29
  - print(sub_arr)
  - print(np_ar1)

  - sub_arr = np.copy(np_ar1[:,1:]) # Correct way

Data Type

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type

Matrix

- We have function for dot product, determinant, transpose, eigen value, solving linear equations, QR decomposition etc.

- Example:
  - mat_a = np.array([[12, 20, 13], [15, 74, 81], [54, 18, 95]])
  - mat_a.T
  - np.linalg.det(mat_a)
  - np.linalg.inv(mat_a)
  - np.trace(mat_a)
  - np.linalg.eig(mat_a)
  - np.linalg.qr(mat_a)
  - np.linalg.solve(A, b) # For solving equation Ax=b
  - np.cross # For cross product

Read File

```
data = np.genfromtxt("Basketball player.txt"
                     , delimiter=","
                     , dtype=None
                     ,encoding="utf-8"
)
```

Exercise