Numpy Tutorial

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

What is an array

An array is a data structure that stores values of same data type. In Python, this is the main difference between arrays and lists. While python lists can contain values corresponding to different data types, arrays in python can only contain values corresponding to same data type.

Why we use array

- Array operations are very very fast.
- Most of the time we use two dimentional array in Exploratory Data Analysis.

In [6]:
```
import numpy as np
```

In [ ]:
```
my_list = [1,2,3,4,5]
arr = np.array(my_list)
```

In [ ]:
```
print(arr)
```

In [36]:
```
my_list1 = (1,2,3,4,5)
my_list2 = (6,7,8,9,10)
my_list3 = (2,4,6,8,10)

arr = np.array([my_list1,my_list2,my_list3])
```

In [37]:
```
arr
```

Out[37]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [ 2,  4,  6,  8, 10]])
```

In [38]:
```
arr.shape
```

Out[38]:
```
(3, 5)
```

In [41]:
```
arr.reshape(5,3)
```

Out[41]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10,  2,  4],
       [ 6,  8, 10]])
```

In [9]:
```
arr
```

Out[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [7]:
```
arr = np.array(range(11))
```

In [ ]:
```

```

In [ ]:
```
arr = np.arrary
```

Indexing

In [49]:
```
arr[1:3,2:4]
```

Out[49]:
```
array([[8, 9],
       [6, 8]])
```

In [50]:
```
arr[1:2,1:-1]
```

Out[50]:
```
array([[7, 8, 9]])
```

In [26]:
```
arr = np.array(range(11))
```

In [27]:
```
arr
```

Out[27]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [12]:
```
arr_1 = arr
arr_1
```

Out[12]:
```
array([  0,   1,   2,   3,   4, 100, 100, 100, 100, 100, 100])
```

In [11]:
```
# Defining the problem
arr_1[5:] = 100 # making change in arr_1
print("arr_1 =",arr_1)    # arr_1 change
print("arr_1 =",arr)      # arr also change
```

```
arr_1 = [  0   1   2   3   4 100 100 100 100 100 100]
arr_1 = [  0   1   2   3   4 100 100 100 100 100 100]
```

In [13]:
```
# Defining solution with copy() method.
arr = np.array(range(11))
print("arr =",arr)
arr_1 = arr.copy() # Making a copy of arr and assigning to arr_1
arr_1[5:] = 100    # Making change in arr_1 but this time not effect on arr
print("arr_1 =",arr_1)
print("arr =",arr)
```

```
arr = [ 0  1  2  3  4  5  6  7  8  9 10]
arr_1 = [  0   1   2   3   4 100 100 100 100 100 100]
arr = [ 0  1  2  3  4  5  6  7  8  9 10]
```

Some Important Conditions used in Exploratory Data Analysis

In [16]:
```
val = 5
arr < 5
```

Out[16]:
```
array([ True,  True,  True,  True,  True, False, False, False, False,
       False, False])
```

In [24]:
```
val = 2
print(arr, 'arr')
print(arr + val, 'arr + val')
print(arr - val, 'arr - val')
print(arr * val, 'arr * val')
print(arr / val, 'arr / val')
print(arr % val, 'arr % val')
print(arr > val, 'arr > val')
print(arr < val, 'arr < val')
# print(arr)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10] arr
[ 2  3  4  5  6  7  8  9 10 11 12] arr + val
[-2 -1  0  1  2  3  4  5  6  7  8] arr - val
[ 0  2  4  6  8 10 12 14 16 18 20] arr * val
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5. ] arr / val
[0 1 0 1 0 1 0 1 0 1 0] arr % val
[False False False  True  True  True  True  True  True  True  True] arr
> val
[ True  True False False False False False False False False False] arr
< val
```

In [35]:
```
print(f"{arr[arr%2 == 0]}\t\t= arr%2 == 0")
print(f"{arr[arr > 2]}\t= arr > 2")
print(f"{arr[arr != 5]}\t= arr != 5")
print(f"{arr[arr == 5]}\t\t\t= arr == 5")
print(arr)
```

```
[ 0  2  4  6  8 10]             = arr%2 == 0
[ 3  4  5  6  7  8  9 10]       = arr > 2
[ 0  1  2  3  4  6  7  8  9 10] = arr != 5
[5]                             = arr == 5
[ 0  1  2  3  4  5  6  7  8  9 10]
```

Applying arithmatic operation between two arrays.

In [39]:
```
array_1 = np.arange(10).reshape(2,5)
print(array_1)
array_2 = np.arange(10).reshape(2,5)
print(array_1)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

In [40]:
```
array_1 * array_2
```

Out[40]:
```
array([[ 0,  1,  4,  9, 16],
       [25, 36, 49, 64, 81]])
```

In [ ]:
```

```

In [56]:
```
arange = np.arange(2,21,2)
```

In [33]:
```
arr
```

Out[33]:
```
array([ 0,  1,  2, 50, 50, 50, 50, 50, 50, 50, 50])
```

In [31]:
```
arr1 = arr[:]
print(arr1)
```

```
[  0   1   2 100 100 100 100 100 100 100 100]
```

In [32]:
```
arr1[3:] = 50
print(arr1)
```

```
[ 0  1  2 50 50 50 50 50 50 50 50]
```

In [29]:
```
#copy function and broadcasting.

arr[3:]=100
```

In [1]:
```
arr
```

```
---------------------------------------------------------------------------
---
NameError                                 Traceback (most recent call la
st)
<ipython-input-1-24a6d41c5b66> in <module>
----> 1 arr

NameError: name 'arr' is not defined
```

Some inbuilt method

arange()

- it is like range() function ### linspace()
- it generate special kind series in which the differnce between all elements is equal. ### copy()
- is used to copy an arry while assigning one arry form one variable to another, this provide solution of ie:.reference type vs value type, by creating a new memory for copied variable.

In [60]:
```
n p.linspace(1,10,50)
```

Out[60]:
```
array([ 1.        ,  1.18367347,  1.36734694,  1.55102041,  1.73469388,
        1.91836735,  2.10204082,  2.28571429,  2.46938776,  2.65306122,
        2.83673469,  3.02040816,  3.20408163,  3.3877551 ,  3.57142857,
        3.75510204,  3.93877551,  4.12244898,  4.30612245,  4.48979592,
        4.67346939,  4.85714286,  5.04081633,  5.2244898 ,  5.40816327,
        5.59183673,  5.7755102 ,  5.95918367,  6.14285714,  6.32653061,
        6.51020408,  6.69387755,  6.87755102,  7.06122449,  7.24489796,
        7.42857143,  7.6122449 ,  7.79591837,  7.97959184,  8.16326531,
        8.34693878,  8.53061224,  8.71428571,  8.89795918,  9.08163265,
```