Basics of NumPy

- NumPy - Introduction and Installation
- NumPy - Arrays Data Structure (1D, 2D, ND arrays)
- Creating Arrays
- NumPy - Data Types
- Array Attributes
- Creating Arrays – Alternative Ways
- Sub-setting, Slicing and Indexing Arrays
- Operations on Arrays
- Array Manipulation

NumPy – Introduction and Installation

- NumPy stands for ‘Numeric Python’
- Used for mathematical and scientific computations
- NumPy array is the most widely used object of the NumPy library

Installing numpy

```
In [ ]: pip install numpy  # Execute this code if import statement gives "ModuleNotFoundErr
```

Importing numpy

```
In [2]: import numpy as np  # Check if library is installed
```

Arrays Data Structure

An `Array` is combination of homogenous data objects and can be indexed across multiple dimensions

Arrays are –

- ordered sequence/collection of Homogenous data
- multidimensional
- mutable

Creating Arrays – From list/tuple

- `np.array()` is used to create a numpy array from a list

Example on 1-D Array

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

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

Example on 2-D Array

```
In [4]: arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
        arr

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

Array Attributes

- Attributes are the features/characteristics of an object that describes the object

- Some of the attributes of the numpy array are:

  - **shape** - Array dimensions
  - **size** - Number of array elements
  - **dtype** - Data type of array elements
  - **ndim** - Number of array dimensions
  - **dtype.name** - Name of data type
  - **astype** - Convert an array to a different type

```
In [5]: arr = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
        arr

Out[5]: array([[ 1,  2,  3,  4],
               [ 5,  6,  7,  8],
               [ 9, 10, 11, 12]])

In [8]: arr.shape

Out[8]: (3, 4)

In [9]: arr.size

Out[9]: 12

In [10]: arr.ndim

Out[10]: 2

In [11]: arr.dtype

Out[11]: dtype('int64')

In [12]: arr.astype(float)
```

```
Out[12]: array([[ 1.,  2.,  3.,  4.],
                [ 5.,  6.,  7.,  8.],
                [ 9., 10., 11., 12.]])
```

Indexing, Slicing and Boolen Indexing

```
In [14]: arr = np.random.randint(5, 50, size = 10)
         arr
```

```
Out[14]: array([24, 25, 26, 19, 41, 38, 29,  8,  8, 28], dtype=int32)
```

1-D Arrays

Ex. Extract first 3 elements

```
In [16]: arr[0:3]
```

```
Out[16]: array([24, 25, 26], dtype=int32)
```

Ex. Extract last 5 elements

```
In [17]: arr[-5 :]
```

```
Out[17]: array([38, 29,  8,  8, 28], dtype=int32)
```

Ex. Extract elements at index position 2, 5, 9.

```
In [18]: arr[[2, 5, 9]]
```

```
Out[18]: array([26, 38, 28], dtype=int32)
```

Ex. Extract elements less than 20

```
In [19]: arr < 20 # Retuns a bool array
```

```
Out[19]: array([False, False, False,  True, False, False, False,  True,  True,
                False])
```

```
In [20]: arr[arr < 20]
```

```
Out[20]: array([19,  8,  8], dtype=int32)
```

2-D Arrays

```
In [21]: arr = np.random.randint(5, 50, size = (6,4))
         arr
```

```
Out[21]: array([[29, 41, 47, 44],
                [16, 14, 37, 29],
                [ 9, 46, 31,  5],
                [17, 11, 11, 22],
                [21,  8, 46, 44],
                [19, 29, 31, 21]], dtype=int32)
```

Ex. Extract first 3 rows

```
In [22]: arr[0:3]
```

```
Out[22]: array([[29, 41, 47, 44],
                [16, 14, 37, 29],
                [ 9, 46, 31,  5]], dtype=int32)
```

Ex. Extract last 2 rows

```
In [23]: arr[-2:]
```

```
Out[23]: array([[21,  8, 46, 44],
                [19, 29, 31, 21]], dtype=int32)
```

Ex. Extract second column

```
In [24]: arr[ :, 2]  # Extracting single column will flatten the array as 1D
```

```
Out[24]: array([47, 37, 31, 11, 46, 31], dtype=int32)
```

Ex. Extract row 2 and 3 and column 2 and 3

```
In [25]: arr[2:4, 2:4]  # Slicing
```

```
Out[25]: array([[31,  5],
                [11, 22]], dtype=int32)
```

```
In [ ]: arr[row : col]
```

Ex. Extract values less than 25

```
In [27]: arr[arr < 25]
```

```
Out[27]: array([16, 14,  9,  5, 17, 11, 11, 22, 21,  8, 19, 21], dtype=int32)
```

Ex. Identify largest value. Extract values less than half of largest values

```
In [32]: arr[arr < np.max(arr)/2]
```

```
Out[32]: array([16, 14,  9,  5, 17, 11, 11, 22, 21,  8, 19, 21], dtype=int32)
```

Examples on Coffee Shop Data Set

Coffee Shop

Mr. Alex owns a huge chain of coffee shop franchises. His franchises are spread across various states in India. There are in all 13 products sold across 156 franchises. Each franchise manager records their monthly sales and profits and compares with its corresponding targeted values. Let us start learning various attributes of numpy and pandas using the Coffee Shop dataset.

Fields in dataset -

- Product Name
- Actual Sales
- Actual Profits
- Targeted Sales
- Targeted Profits

Ex. Create an array of coffee products, sales and profits

In [33]:
```
coffee_products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling',
sales = np.array( [52248.0, 14068.0, 71060.0, 60014.0, 69925.0, 27711.0, 19231.0, 2
profits = np.array([17444.0, 5041.0, 28390.0, 20459.0, 23432.0, 7691.0, -2954.0, 71
```

Ex. How many products are there in the dataset?

In [34]:
```
coffee_products.size
```

Out[34]:
```
10
```

Ex. Sales greater than 50,000

In [35]:
```
sales[sales > 50000]
```

Out[35]:
```
array([52248., 71060., 60014., 69925.])
```

Ex. Identify Losses

In [36]:
```
profits[profits < 0 ]
```

Out[36]:
```
array([-2954.])
```

Ex. Products in loss

In [37]:
```
coffee_products[profits< 0]
```

Out[37]:
```
array(['Green Tea'], dtype='<U17')
```

Ex. Product with maximum Sales

In [39]:
```
coffee_products[sales == max(sales)][0]
```

Out[39]:
```
np.str_('Colombian')
```

Array Operations

Arithmetic operations on Arrays -

- Addition, Substraction, Multiplication, Division, etc.
- Operations on array and a scalar value
- Operations between two arrays
- Matrix Operations - Multiplication(np.dot()), Transpose(np.transpose())

Array and Scalar

In [40]:

```
arr1 = np.random.randint(1,10,size = 5)
arr1
```

Out[40]:

```
array([7, 1, 8, 2, 6], dtype=int32)
```

In [41]:

```
arr1 + 5 # Addition
```

Out[41]:

```
array([12,  6, 13,  7, 11], dtype=int32)
```

In [42]:

```
arr1 * 1.07
```

Out[42]:

```
array([7.49, 1.07, 8.56, 2.14, 6.42])
```

In [43]:

```
arr1 - 5 # Substraction
```

Out[43]:

```
array([ 2, -4,  3, -3,  1], dtype=int32)
```

In [44]:

```
arr1 * 5 # Multiplication
```

Out[44]:

```
array([35,  5, 40, 10, 30], dtype=int32)
```

In [45]:

```
arr1 / 5 # Division
```

Out[45]:

```
array([1.4, 0.2, 1.6, 0.4, 1.2])
```

In [46]:

```
arr1 // 5 # Floor Division
```

Out[46]:

```
array([1, 0, 1, 0, 1], dtype=int32)
```

In [47]:

```
arr1 % 5 # Modulus
```

Out[47]:

```
array([2, 1, 3, 2, 1], dtype=int32)
```

Two Arrays

In [48]:

```
arr1 = np.random.randint(1,10,size = 5)
arr1
```

```
Out[48]: array([6, 8, 6, 2, 2], dtype=int32)
```

```
In [49]: arr2 = np.random.randint(1,10,size = 5)
         arr2
```

```
Out[49]: array([2, 8, 3, 2, 6], dtype=int32)
```

```
In [50]: arr1 + arr2 # Addition
```

```
Out[50]: array([ 8, 16,  9,  4,  8], dtype=int32)
```

```
In [51]: arr1 - arr2 # Substraction
```

```
Out[51]: array([ 4,  0,  3,  0, -4], dtype=int32)
```

```
In [52]: arr1 * arr2 # Multiplication
```

```
Out[52]: array([12, 64, 18,  4, 12], dtype=int32)
```

```
In [53]: arr1 / arr2 # Division
```

```
Out[53]: array([3.        , 1.        , 2.        , 1.        , 0.33333333])
```

```
In [54]: arr1 // arr2 # Floor Division
```

```
Out[54]: array([3, 1, 2, 1, 0], dtype=int32)
```

```
In [55]: arr1 % arr2 # Modulus
```

```
Out[55]: array([0, 0, 0, 0, 2], dtype=int32)
```

Relational operations on Arrays -

- ==, !=, <, >, <=, >=
- Operations on array and a scalar value
- Operations between two arrays

Array and Scalar

```
In [56]: arr1 = np.random.randint(1,10,size = 5)
         arr1
```

```
Out[56]: array([8, 3, 2, 3, 1], dtype=int32)
```

```
In [57]: arr1 == 5
```

```
Out[57]: array([False, False, False, False, False])
```

```
In [58]: arr1 != 5
```

```
Out[58]: array([ True,  True,  True,  True,  True])

In [59]: arr1 < 5

Out[59]: array([False,  True,  True,  True,  True])

In [60]: arr1 > 5

Out[60]: array([ True, False, False, False, False])

In [61]: arr1 <= 5

Out[61]: array([False,  True,  True,  True,  True])

In [62]: arr1 >= 5

Out[62]: array([ True, False, False, False, False])
```

Two Arrays

```
In [63]: arr1 = np.random.randint(1,10,size = 5)
         arr1

Out[63]: array([2, 3, 3, 5, 5], dtype=int32)

In [64]: arr2 = np.random.randint(1,10,size = 5)
         arr2

Out[64]: array([9, 7, 2, 8, 4], dtype=int32)

In [65]: arr1 == arr2

Out[65]: array([False, False, False, False, False])

In [66]: arr1 != arr2

Out[66]: array([ True,  True,  True,  True,  True])

In [67]: arr1 < arr2

Out[67]: array([ True,  True, False,  True, False])

In [68]: arr1 > arr2

Out[68]: array([False, False,  True, False,  True])

In [69]: arr1 <= arr2

Out[69]: array([ True,  True, False,  True, False])

In [70]: arr1 >= arr2
```

```
Out[70]: array([False, False,  True, False,  True])
```

Logical operations on Arrays -

- np.logical_or()
- np.logical_and()
- np.logical_not()
- np.logical_xor()

```
In [71]: arr1 = np.random.randint(1,10,size = 5)
         arr1
```

```
Out[71]: array([6, 3, 5, 1, 2], dtype=int32)
```

```
In [72]: arr2 = np.random.randint(1,10,size = 5)
         arr2
```

```
Out[72]: array([9, 8, 4, 9, 3], dtype=int32)
```

```
In [73]: np.logical_and(arr1 > 5, arr2 > 5)
```

```
Out[73]: array([ True, False, False, False, False])
```

```
In [74]: np.logical_or(arr1 > 5, arr2 > 5)
```

```
Out[74]: array([ True,  True, False,  True, False])
```

```
In [75]: np.logical_not(arr1 > 5)
```

```
Out[75]: array([False,  True,  True,  True,  True])
```

```
In [76]: np.logical_xor(arr1 > 5, arr2 > 5)
```

```
Out[76]: array([False,  True, False,  True, False])
```

```
In [77]: arr1 > 5
```

```
Out[77]: array([ True, False, False, False, False])
```

```
In [78]: arr2 > 5
```

```
Out[78]: array([ True,  True, False,  True, False])
```

Set Operations on Arrays

Applicable to 1-D Ararys only

- np.unique() - Find the unique elements of an array.
- np.isin() - Test whether each element of a 1-D array is also present in a second array.
- np.intersect1d() - Find the intersection of two arrays.

- np.setdiff1d() - Find the set difference of two arrays.
- np.union1d() - Find the union of two arrays.

```
In [81]: arr1 = np.random.randint(1,5,size = 10)
         arr1

Out[81]: array([2, 3, 1, 2, 1, 4, 1, 3, 4, 2], dtype=int32)
```

```
In [82]: np.unique(arr1)

Out[82]: array([1, 2, 3, 4], dtype=int32)
```

```
In [83]: np.unique(arr1, return_index= True, return_counts= True)

Out[83]: (array([1, 2, 3, 4], dtype=int32), array([2, 0, 1, 5]), array([3, 3, 2, 2]))
```

```
In [88]: arr1 = np.random.randint(1,10,size = 10)
         arr1

Out[88]: array([5, 4, 5, 3, 3, 8, 6, 8, 4, 7], dtype=int32)
```

```
In [89]: arr2 = np.random.randint(1,5,size = 10)
         arr2

Out[89]: array([4, 2, 3, 3, 4, 1, 1, 1, 1, 3], dtype=int32)
```

```
In [90]: np.isin(arr1, arr2)

Out[90]: array([False,  True, False,  True,  True, False, False, False,  True,
                False])
```

```
In [91]: np.intersect1d(arr1, arr2)

Out[91]: array([3, 4], dtype=int32)
```

```
In [92]: np.setdiff1d(arr1, arr2)

Out[92]: array([5, 6, 7, 8], dtype=int32)
```

```
In [93]: np.union1d(arr1, arr2)

Out[93]: array([1, 2, 3, 4, 5, 6, 7, 8], dtype=int32)
```

Array Functions/Methods

- np.all(), np.any()
- arr.sum()
- arr.min(), arr.max(), arr.argmin(), arr.agrmax()
- np.round()
- np.mean(), np.median(), np.average(), np.percentile()

```
In [94]: np.all(np.isin(arr1, arr2))  # There are some elements in arr1 which are not presen
```

```
Out[94]: np.False_
```

```
In [96]: np.any(np.isin(arr1, arr2))  # There is atleast 1 element in arr1 which is present
```

```
Out[96]: np.True_
```

```
In [97]: arr1.sum()
```

```
Out[97]: np.int64(53)
```

```
In [98]: np.sum(arr1)
```

```
Out[98]: np.int64(53)
```

```
In [100… np.argmax(arr1)  # index position of largest element
```

```
Out[100… np.int64(5)
```

Ex. Product with maximum sales?

```
In [104… coffee_products[sales == sales.max()]  # Boolean/conditional indexing - returns an
```

```
Out[104… array(['Colombian'], dtype='<U17')
```

```
In [102… coffee_products[sales.argmax()]  # Normal Indexing - returns the object at the inde
```

```
Out[102… np.str_('Colombian')
```

Examples on Coffee Shop Data Set

Data

```
In [105… import numpy as np
         products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling', 'Decaf
         sales = np.array([52248.0, 14068.0, 71060.0, 60014.0, 69925.0, 27711.0, 19231.0, 24
         profits = np.array([17444.0, 5041.0, 28390.0, 20459.0, 23432.0, 7691.0, -2954.0, 71
         target_profits = np.array([15934.0, 4924.0, 31814.0, 19649.0, 24934.0, 8461.0, 7090
         target_sales = np.array([48909.0, 13070.0, 80916.0, 57368.0, 66906.0, 30402.0, 1821
```

Ex. Identify the products meeting the Target Profits

```
In [109… profit_ach = products[profits >= target_profits]
         profit_ach
```

```
Out[109… array(['Caffe Latte', 'Cappuccino', 'Darjeeling', 'Lemon', 'Mint'],
               dtype='<U17')
```

Ex. Are the above products meeting their sales target too? Yes/No.

In [110…

```
sales_ach = products[sales >= target_sales]
if np.all(np.isin(profit_ach, sales_ach)) :
    print("Yes")
else :
    print("No")
```

```
Yes
```

Ex. Identify products meeting sales targets but not profit targets

In [111…

```
np.all(np.isin(sales_ach, profit_ach))  # There are some products which are achievi
```

Out[111…

```
np.False_
```

In [112…

```
np.setdiff1d(sales_ach, profit_ach)
```

Out[112…

```
array(['Decaf Irish Cream', 'Green Tea', 'Regular Espresso'], dtype='<U17')
```

Sorting Arrays

In [114…

```
prices = np.random.randint(1000, 50000, 10)
prices
```

Out[114…

```
array([38484, 10175, 27362, 39338, 46416,  5660,  3298, 42721, 23197,
       21773], dtype=int32)
```

In [115…

```
np.sort(prices)  # Returns a new array object by sorting the elements
```

Out[115…

```
array([ 3298,  5660, 10175, 21773, 23197, 27362, 38484, 39338, 42721,
       46416], dtype=int32)
```

In [116…

```
prices.sort()  # modify the original array
```

Ex. Sort the products in DESC order of their prices

In [123…

```
products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling', 'Decaf
prices = np.random.randint(1000, 50000, 10)
prices
```

Out[123…

```
array([36668, 33979, 15292, 16502, 47834, 19283, 24845, 12767, 42152,
       38761], dtype=int32)
```

In [124…

```
np.argsort(prices)  # Returns the index position of elements from prices array afte
```

Out[124…

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

In [125…

```
sort_order = np.argsort(prices)[::-1]  # DESC order
```

In [126…

```
products[sort_order]  # Sorting the products by sort_order
```

Out[126…

```
array(['Decaf Irish Cream', 'Mint', 'Regular Espresso', 'Caffe Latte',
       'Cappuccino', 'Green Tea', 'Earl Grey', 'Darjeeling', 'Colombian',
       'Lemon'], dtype='<U17')
```

Array Manipulations

- **Changing Shape** – np.reshape()
- **Adding/Removing Elements** – np.append(), np.insert(), np.delete()
- **Splitting Arrays** – np.hsplit(), np.vsplit(), arr_obj.flatten()
- **Sorting Arrays** - arr_obj.sort(), arr_obj.argsort()

In [131…

```
arr = np.arange(1, 13)
arr
```

Out[131…

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

np.reshape()

In [132…

```
arr = np.reshape(arr, (4, 3))
arr
```

Out[132…

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

np.append()

In [133…

```
np.append(arr, 20) # Flattens the array
```

Out[133…

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 20])
```

In [137…

```
# insert a row [10, 20, 30] at the end
row = np.reshape(np.array([10, 20, 30]), (1, 3))
np.append(arr, row, axis=0)
```

Out[137…

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [10, 20, 30]])
```

In [139…

```
# insert a row [10, 20, 30] at the end
col = np.reshape(np.array([10, 20, 30, 40]), (4, 1))
np.append(arr, col, axis=1)
```

Out[139…

```
array([[ 1,  2,  3, 10],
       [ 4,  5,  6, 20],
       [ 7,  8,  9, 30],
       [10, 11, 12, 40]])
```

np.insert()

In [140…

```
arr = np.reshape(np.arange(1,13), (4,3))
arr
```

Out[140…

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [141…

```
np.insert(arr, 1, 5) # Flattens the arr and inserts 5 at index 1
```

Out[141…

```
array([ 1,  5,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [142…

```
np.insert(arr, 1, 5, axis=0) # Inserts [5, 5, 5] as row 1
```

Out[142…

```
array([[ 1,  2,  3],
       [ 5,  5,  5],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [143…

```
np.insert(arr, 1, 5, axis=1) # Inserts [5, 5, 5, 5] as column 1
```

Out[143…

```
array([[ 1,  5,  2,  3],
       [ 4,  5,  5,  6],
       [ 7,  5,  8,  9],
       [10,  5, 11, 12]])
```

In [144…

```
np.insert(arr, 1, [10, 20, 30, 40], axis=1)
```

Out[144…

```
array([[ 1, 10,  2,  3],
       [ 4, 20,  5,  6],
       [ 7, 30,  8,  9],
       [10, 40, 11, 12]])
```

In [145…

```
np.insert(arr, 1, [10, 20, 30], axis=0)
```

Out[145…

```
array([[ 1,  2,  3],
       [10, 20, 30],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

np.delete()

In [146…

```
arr = np.reshape(np.arange(1,13), (4,3))
arr
```

Out[146…

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [147…

```
np.delete(arr, 1) # Flattens the arr and deletes element at index 1
```

Out[147…

```
array([ 1,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [148…

```
np.delete(arr, 1, axis=0) # deletes row 1
```

Out[148…

```
array([[ 1,  2,  3],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [149…

```
np.delete(arr, 1, axis=1) # deletes column 1
```

Out[149…

```
array([[ 1,  3],
       [ 4,  6],
       [ 7,  9],
       [10, 12]])
```

In [150…

```
np.delete(arr,[0,2], axis=0) # deletes selected rows
```

Out[150…

```
array([[ 4,  5,  6],
       [10, 11, 12]])
```

In [151…

```
np.delete(arr,[0,2], axis=1) # deletes selected columns
```

Out[151…

```
array([[ 2],
       [ 5],
       [ 8],
       [11]])
```

np.hsplit(), np.vsplit()

In [152…

```
arr = np.reshape(np.arange(1, 25), (6,4))
arr
```

Out[152…

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```

In [153…

```
np.vsplit(arr, 2)  # split on row
```

Out[153…

```
[array([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]]),
 array([[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]])]
```

In [154…

```
np.hsplit(arr, 2) # split on columns
```

```
Out[154…  [array([[ 1,  2],
                  [ 5,  6],
                  [ 9, 10],
                  [13, 14],
                  [17, 18],
                  [21, 22]]),
           array([[ 3,  4],
                  [ 7,  8],
                  [11, 12],
                  [15, 16],
                  [19, 20],
                  [23, 24]])]
```

flatten()

```
In [155…  arr
```

```
Out[155…  array([[ 1,  2,  3,  4],
                 [ 5,  6,  7,  8],
                 [ 9, 10, 11, 12],
                 [13, 14, 15, 16],
                 [17, 18, 19, 20],
                 [21, 22, 23, 24]])
```

```
In [157…  arr.flatten(order = "C") # returns a 1-D array
```

```
Out[157…  array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
                 18, 19, 20, 21, 22, 23, 24])
```

```
In [158…  arr.flatten(order = "F") # returns a 1-D array
```

```
Out[158…  array([ 1,  5,  9, 13, 17, 21,  2,  6, 10, 14, 18, 22,  3,  7, 11, 15, 19,
                 23,  4,  8, 12, 16, 20, 24])
```

```
In [ ]:  help(arr.flatten)
```

Ex. Find all products which are not achieveing their profit targets. Calculate percent target achieved

```
In [159…  prod = coffee_products[profits < target_profits]
          prof = profits[profits < target_profits]
          tar_prof = target_profits[profits < target_profits]
```

```
In [161…  800/1000 * 100
```

```
Out[161…  80.0
```

```
In [164…  percentages = np.round(prof/tar_prof * 100, 2)
          percentages
```

```
Out[164…  array([ 89.24,  93.98,  90.9 , -41.66,  91.39])
```

```
In [166…  # Replace negative values with zero
          percentages[percentages < 0 ] = 0
          percentages
```

```
Out[166…   array([89.24, 93.98, 90.9 ,  0.  , 91.39])
```