TASK #1: DEFINE SINGLE AND MULTI-DIMENSIONAL NUMPY ARRAYS

In [1]:
```
# NumPy is a Linear Algebra Library used for multidimensional arrays
# NumPy brings the best of two worlds: (1) C/Fortran computational efficiency, (2) Python language easy syntax
import numpy as np

# Let's define a one-dimensional array
list_1 = [50, 60, 80, 100, 200, 300, 500, 600]
list_1
```

Out[1]:
```
[50, 60, 80, 100, 200, 300, 500, 600]
```

In [2]:
```
# Let's create a numpy array from the list "my_list"
my_numpy_array = np.array(list_1)
my_numpy_array
```

Out[2]:
```
array([ 50,  60,  80, 100, 200, 300, 500, 600])
```

In [3]:
```
type(my_numpy_array)
```

Out[3]:
```
numpy.ndarray
```

In [4]:
```
# Multi-dimensional (Matrix definition)
my_matrix = np.array([[2, 5, 8], [7, 3, 6]])
my_matrix
```

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

MINI CHALLENGE #1:

- Write a code that creates the following 2x4 numpy array

```
[[3 7 9 3]
[4 3 2 2]]
```

In [ ]:
```

```

TASK #2: LEVERAGE NUMPY BUILT-IN METHODS AND FUNCTIONS

In [6]:
```
# "rand()" uniform distribution between 0 and 1
x = np.random.rand(20)
x
```

Out[6]:
```
array([0.24550682, 0.92290457, 0.4982003 , 0.44254441, 0.64748482,
       0.07986556, 0.70048155, 0.0986679 , 0.79340111, 0.31828063,
       0.54551631, 0.61038962, 0.15794414, 0.17205903, 0.66409968,
       0.4449435 , 0.03330255, 0.46514692, 0.40425268, 0.49549523])
```

In [7]:
```
# you can create a matrix of random number as well
x = np.random.rand(3, 3)
x
```

Out[7]:
```
array([[0.39727509, 0.98642314, 0.83894724],
       [0.96241275, 0.00115693, 0.05789424],
       [0.34128873, 0.09261359, 0.26125384]])
```

In [9]:
```
# "randint" is used to generate random integers between upper and lower bounds
x = np.random.randint(1, 50)
x
```

Out[9]:
```
15
```

In [10]:
```
# "randint" can be used to generate a certain number of random itegers as follows
x = np.random.randint(1, 100, 15)
x
```

Out[10]:
```
array([15,  6, 69,  8, 71, 53, 52, 42, 18, 56, 98, 50, 83, 31, 10])
```

In [11]:
```
# np.arange creates an evenly spaced values within a given interval
x = np.arange(1, 50)
x
```

Out[11]:
```
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, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

In [12]:
```
# create a diagonal of ones and zeros everywhere else
x = np.eye(7)
x
```

Out[12]:
```
array([[1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1.]])
```

In [13]:
```
# Matrix of ones
x = np.ones((7, 7))
x
```

Out[13]:
```
array([[1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
```

In [14]:
```
# Array of zeros
x = np.zeros(8)
x
```

Out[14]:
```
array([0., 0., 0., 0., 0., 0., 0., 0.])
```

MINI CHALLENGE #2:

- Write a code that takes in a positive integer "x" from the user and creates a 1x10 array with random numbers ranging from 0 to "x"

In [ ]:
```

```

TASK #3: PERFORM MATHEMATICAL OPERATIONS IN NUMPY

In [16]:
```
# np.arange() returns an evenly spaced values within a given interval
x = np.arange(1, 10)
x
```

Out[16]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [17]:
```
y = np.arange(1, 10)
y
```

Out[17]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [18]:
```
# Add 2 numpy arrays together
sum = x + y
sum
```

Out[18]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [19]:
```
squared = x ** 2
squared
```

Out[19]:
```
array([ 1,  4,  9, 16, 25, 36, 49, 64, 81], dtype=int32)
```

In [20]:
```
sqrt = np.sqrt(squared)
sqrt
```

Out[20]:
```
array([1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

In [21]:
```
z = np.exp(y)
z
```

Out[21]:
```
array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
       1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
       8.10308393e+03])
```

MINI CHALLENGE #3:

- Given the X and Y values below, obtain the distance between them

```
X = [5, 7, 20]
Y = [9, 15, 4]
```

In [ ]:
```

```

TASK #4: PERFORM ARRAYS SLICING AND INDEXING

In [26]:
```
my_numpy_array = np.array([3, 5, 6, 2, 8, 10, 20, 50])
my_numpy_array
```

Out[26]:
```
array([ 3,  5,  6,  2,  8, 10, 20, 50])
```

In [27]:
```
# Access specific index from the numpy array
my_numpy_array[0]
```

Out[27]:
```
3
```

In [28]:
```
# Starting from the first index 0 up until and NOT including the last element
my_numpy_array[0:3]
```

Out[28]:
```
array([3, 5, 6])
```

In [29]:
```
# Broadcasting, altering several values in a numpy array at once
my_numpy_array[0:4] = 7
my_numpy_array
```

Out[29]:
```
array([ 7,  7,  7,  7,  8, 10, 20, 50])
```

In [30]:
```
# Let's define a two dimensional numpy array
matrix = np.random.randint(1, 10, (4, 4))
matrix
```

Out[30]:
```
array([[6, 9, 7, 1],
       [6, 5, 5, 5],
       [6, 9, 1, 8],
       [8, 9, 3, 1]])
```

In [31]:
```
# Get a row from a mtrix
matrix[2]
```

Out[31]:
```
array([6, 9, 1, 8])
```

In [32]:
```
# Get one element
matrix[0][2]
```

Out[32]:
```
7
```

MINI CHALLENGE #4:

- In the following matrix, replace the last row with 0

```
X = [2 30 20 -2 -4]
    [3 4  40 -3 -2]
    [-3 4 -6 90 10]
    [25 45 34 22 12]
    [13 24 22 32 37]
```

In [ ]:
```

```

TASK #5: PERFORM ELEMENTS SELECTION (CONDITIONAL)

In [42]:
```
matrix = np.random.randint(1,10, (5, 5))
matrix
```

Out[42]:
```
array([[4, 4, 3, 9, 6],
       [9, 9, 8, 9, 2],
       [4, 2, 1, 3, 3],
       [8, 1, 1, 6, 3],
       [1, 3, 6, 8, 8]])
```

In [43]:
```
new_matrix = matrix[matrix>7]
new_matrix
```

Out[43]:
```
array([9, 9, 9, 8, 9, 8, 8, 8])
```

In [44]:
```
# Obtain odd elements only
new_matrix = matrix[matrix % 2 == 1]
new_matrix
```

Out[44]:
```
array([3, 9, 9, 9, 9, 1, 3, 3, 1, 1, 3, 1, 3])
```

MINI CHALLENGE #5:

- In the following matrix, replace negative elements by 0 and replace odd elements with -2

```
X = [2 30 20 -2 -4]
    [3 4  40 -3 -2]
    [-3 4 -6 90 10]
    [25 45 34 22 12]
    [13 24 22 32 37]
```

In [ ]:
```

```

TASK #6: UNDERSTAND PANDAS FUNDAMENTALS

In [ ]:
```
# Pandas is a data manipulation and analysis tool that is built on Numpy.
# Pandas uses a data structure known as DataFrame (think of it as Microsoft excel in Python).
# DataFrames empower programmers to store and manipulate data in a tabular fashion (rows and columns).
# Series Vs. DataFrame? Series is considered a single column of a DataFrame.
```

In [45]:
```
import pandas as pd
```

In [53]:
```
# Let's define a two-dimensional Pandas DataFrame
# Note that you can create a pandas dataframe from a python dictionary
bank_client_df = pd.DataFrame({'Bank client ID':[111, 222, 333, 444],
                               'Bank Client Name':['Chanel', 'Steve', 'Mitch', 'Ryan'],
                               'Net worth [$]':[3500, 29000, 10000, 2000],
                               'Years with bank':[3, 4, 9, 5]})
bank_client_df
```

Out[53]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

In [54]:
```
# Let's obtain the data type
type(bank_client_df)
```

Out[54]:
```
pandas.core.frame.DataFrame
```

In [55]:
```
# you can only view the first couple of rows using .head()
bank_client_df.head(2)
```

Out[55]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4

In [56]:
```
# you can only view the last couple of rows using .tail()
bank_client_df.tail(1)
```

Out[56]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
3	444	Ryan	2000	5

MINI CHALLENGE #6:

- A porfolio contains a collection of securities such as stocks, bonds and ETFs. Define a dataframe named 'portfolio_df' that holds 3 different stock ticker symbols, number of shares, and price per share (feel free to choose any stocks)
- Calculate the total value of the porfolio including all stocks

In [ ]:
```

```

TASK #7: PANDAS WITH CSV AND HTML DATA

In [62]:
```
# Pandas is used to read a csv file and store data in a DataFrame
bank_df = pd.read_csv('bank_client_information.csv')
```

In [63]:
```
bank_df
```

Out[63]:

	First Name	Last Name	Email	Postal Code	Net Worth
0	Joseph	Patton	daafeja@boh.jm	M6U 5U7	$2,629.13
1	Noah	Moran	guutodi@bigwoc.kw	K2D 4M9	$8,626.96
2	Nina	Keller	azikez@gahew.mr	S1T 4E6	$9,072.02

In [64]:
```
# Read tabular data using read_html
house_prices_df = pd.read_html('https://www.livingin-canada.com/house-prices-canada.html')
```

In [65]:
```
house_prices_df[0]
```

Out[65]:

	City	Average House Price	12 Month Change
0	Vancouver, BC	$1,036,000	+ 2.63 %
1	Toronto, Ont	$870,000	+10.2 %
2	Ottawa, Ont	$479,000	+ 15.4 %
3	Calgary, Alb	$410,000	– 1.5 %
4	Montreal, Que	$435,000	+ 9.3 %
5	Halifax, NS	$331,000	+ 3.6 %
6	Regina, Sask	$254,000	– 3.9 %
7	Fredericton, NB	$198,000	– 4.3 %
8	(adsbygoogle = window.adsbygoogle || []).push(...	(adsbygoogle = window.adsbygoogle || []).push(...	(adsbygoogle = window.adsbygoogle || []).push(...

In [66]:
```
house_prices_df[1]
```

Out[66]:

	Province	Average House Price	12 Month Change
0	British Columbia	$736,000	+ 7.6 %
1	Ontario	$594,000	– 3.2 %
2	Alberta	$353,000	– 7.5 %
3	Quebec	$340,000	+ 7.6 %
4	Manitoba	$295,000	– 1.4 %
5	Saskatchewan	$271,000	– 3.8 %
6	Nova Scotia	$266,000	+ 3.5 %
7	Prince Edward Island	$243,000	+ 3.0 %
8	Newfoundland / Labrador	$236,000	– 1.6 %
9	New Brunswick	$183,000	– 2.2 %
10	Canadian Average	$488,000	– 1.3 %
11	(adsbygoogle = window.adsbygoogle || []).push(...	(adsbygoogle = window.adsbygoogle || []).push(...	(adsbygoogle = window.adsbygoogle || []).push(...

MINI CHALLENGE #7:

- Write a code that uses Pandas to read tabular US retirement data
- You can use data from here: https://www.ssa.gov/oact/progdata/nra.html (https://www.ssa.gov/oact/progdata/nra.html)

In [ ]:
```

```

TASK #8: PANDAS OPERATIONS

In [67]:
```
# Let's define a dataframe as follows:
bank_client_df = pd.DataFrame({'Bank client ID':[111, 222, 333, 444],
                               'Bank Client Name':['Chanel', 'Steve', 'Mitch', 'Ryan'],
                               'Net worth [$]':[3500, 29000, 10000, 2000],
                               'Years with bank':[3, 4, 9, 5]})
bank_client_df
```

Out[67]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

In [68]:
```
# Pick certain rows that satisfy a certain criteria
df_loyal = bank_client_df[ (bank_client_df['Years with bank'] >= 5) ]
df_loyal
```

Out[68]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
2	333	Mitch	10000	9
3	444	Ryan	2000	5

In [69]:
```
# Delete a column from a DataFrame
del bank_client_df['Bank client ID']
bank_client_df
```

Out[69]:

	Bank Client Name	Net worth [$]	Years with bank
0	Chanel	3500	3
1	Steve	29000	4
2	Mitch	10000	9
3	Ryan	2000	5

MINI CHALLENGE #8:

- Using "bank_client_df" DataFrame, leverage pandas operations to only select high networth individuals with minimum $5000
- What is the combined networth for all customers with 5000+ networth?

In [ ]:
```

```

TASK #9: PANDAS WITH FUNCTIONS

In [71]:
```
# Let's define a dataframe as follows:
bank_client_df = pd.DataFrame({'Bank client ID':[111, 222, 333, 444],
                               'Bank Client Name':['Chanel', 'Steve', 'Mitch', 'Ryan'],
                               'Net worth [$]':[3500, 29000, 10000, 2000],
                               'Years with bank':[3, 4, 9, 5]})
bank_client_df
```

Out[71]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

In [73]:
```
# Define a function that increases all clients networth (stocks) by a fixed value of 20% (for simplicity sake)
def networth_update(balance):
    return balance * 1.2 # assume that stock prices increased by 20%
```

In [74]:
```
# You can apply a function to the DataFrame
bank_client_df['Net worth [$]'].apply(networth_update)

```

Out[74]:
```
0     4200.0
1    34800.0
2    12000.0
3     2400.0
Name: Net worth [$], dtype: float64
```

In [75]:
```
bank_client_df['Bank Client Name'].apply(len)
```

Out[75]:
```
0    6
1    5
2    5
3    4
Name: Bank Client Name, dtype: int64
```

MINI CHALLENGE #9:

- Define a function that triples the stock prices and adds $200
- Apply the function to the DataFrame
- Calculate the updated total networth of all clients combined

In [ ]:
```

```

TASK #10: PERFORM SORTING AND ORDERING IN PANDAS

In [80]:
```
# Let's define a dataframe as follows:
bank_client_df = pd.DataFrame({'Bank client ID':[111, 222, 333, 444],
                               'Bank Client Name':['Chanel', 'Steve', 'Mitch', 'Ryan'],
                               'Net worth [$]':[3500, 29000, 10000, 2000],
                               'Years with bank':[3, 4, 9, 5]})
bank_client_df
```

Out[80]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

In [81]:
```
# You can sort the values in the dataframe according to number of years with bank
bank_client_df.sort_values(by = 'Years with bank')
```

Out[81]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
3	444	Ryan	2000	5
2	333	Mitch	10000	9

In [82]:
```
# Note that nothing changed in memory! you have to make sure that inplace is set to True
bank_client_df
```

Out[82]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
2	333	Mitch	10000	9
3	444	Ryan	2000	5

In [83]:
```
# Set inplace = True to ensure that change has taken place in memory
bank_client_df.sort_values(by = 'Years with bank', inplace = True)
```

In [84]:
```
# Note that now the change (ordering) took place
bank_client_df
```

Out[84]:

	Bank client ID	Bank Client Name	Net worth [$]	Years with bank
0	111	Chanel	3500	3
1	222	Steve	29000	4
3	444	Ryan	2000	5
2	333	Mitch	10000	9

TASK #11: PERFORM CONCATENATING AND MERGING WITH PANDAS

In [90]:
```
# Check this out: https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html
```

In [91]:
```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
index=[0, 1, 2, 3])
```

In [92]:
```
df1
```

Out[92]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [93]:
```
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
index=[4, 5, 6, 7])
```

In [94]:
```
df2
```

Out[94]:

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

In [95]:
```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
index=[8, 9, 10, 11])
```

In [96]:
```
df3
```

Out[96]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

In [97]:
```
pd.concat([df1, df2, df3])
```

Out[97]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

TASK #12: PROJECT AND CONCLUDING REMARKS

- Define a dataframe named 'Bank_df_1' that contains the first and last names for 5 bank clients with IDs = 1, 2, 3, 4, 5
- Assume that the bank got 5 new clients, define another dataframe named 'Bank_df_2' that contains a new clients with IDs = 6, 7, 8, 9, 10
- Let's assume we obtained additional information (Annual Salary) about all our bank customers (10 customers)
- Concatenate both 'bank_df_1' and 'bank_df_2' dataframes
- Merge client names and their newly added salary information using the 'Bank Client ID'
- Let's assume that you became a new client to the bank
- Define a new DataFrame that contains your information such as client ID (choose 11), first name, last name, and annual salary.
- Add this new dataframe to the original dataframe 'bank_df_all'.

In [ ]:
```
```

EXCELLENT JOB!

MINI CHALLENGES SOLUTIONS

MINI CHALLENGE #1 SOLUTION: