# Python for Data Science : Week 2 - Tutorial

# Introduction to Data Science using Python

Sequence Data Operations in Data Science using Python

Data Science is a multidisciplinary field that involves extracting insights and knowledge from data. One of the fundamental aspects of data science is handling and analyzing sequence data, such as lists, strings, time series, and more. Python provides powerful tools, especially through libraries like NumPy and pandas, to perform various sequence data operations efficiently. In this note, we will explore some essential sequence data operations in Python for data science.

1. List Operations
2. String Operations
3. Time Series Operations

In [124]:

```
import numpy as np
```

Example 1: Creating a 1D Array

In [125]:

```
# Create a 1D NumPy array
arr1d = np.array([1, 2, 3, 4, 5]) #defining an 1d array
print(arr1d)
```

```
[1 2 3 4 5]
```

Example 2: Creating a 2D Array

In [127]:

```
# Create a 2D NumPy array
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d)
```

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

In [128]:

```
arrex = np.array([[1,2,10],[3,4,12]])
print(arrex)
```

```
[[ 1  2 10]
 [ 3  4 12]]
```

Example 3: Array Shape and Dimensions

In [131]:

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr)
print(arr.shape)  #shape of the array
print(arr.ndim)   #dimension

shape = np.shape(arr)
print(shape)
```

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

Example 4: Element-wise Operations

In [132]:

```
arr1 = np.array([1, 2, 3]) #array 1
arr2 = np.array([4, 5, 6]) #array 2

result = arr1 + arr2  #sum opertaion
print(result)
```

```
[5 7 9]
```

Example 5: Mathematical Functions

In [13]:

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

# Square each element of the array
squared = np.square(arr)
print(squared)

# Calculate the mean of the array
mean_val = np.mean(arr)  #mean value
print(mean_val)

#np.sum
#np.std
```

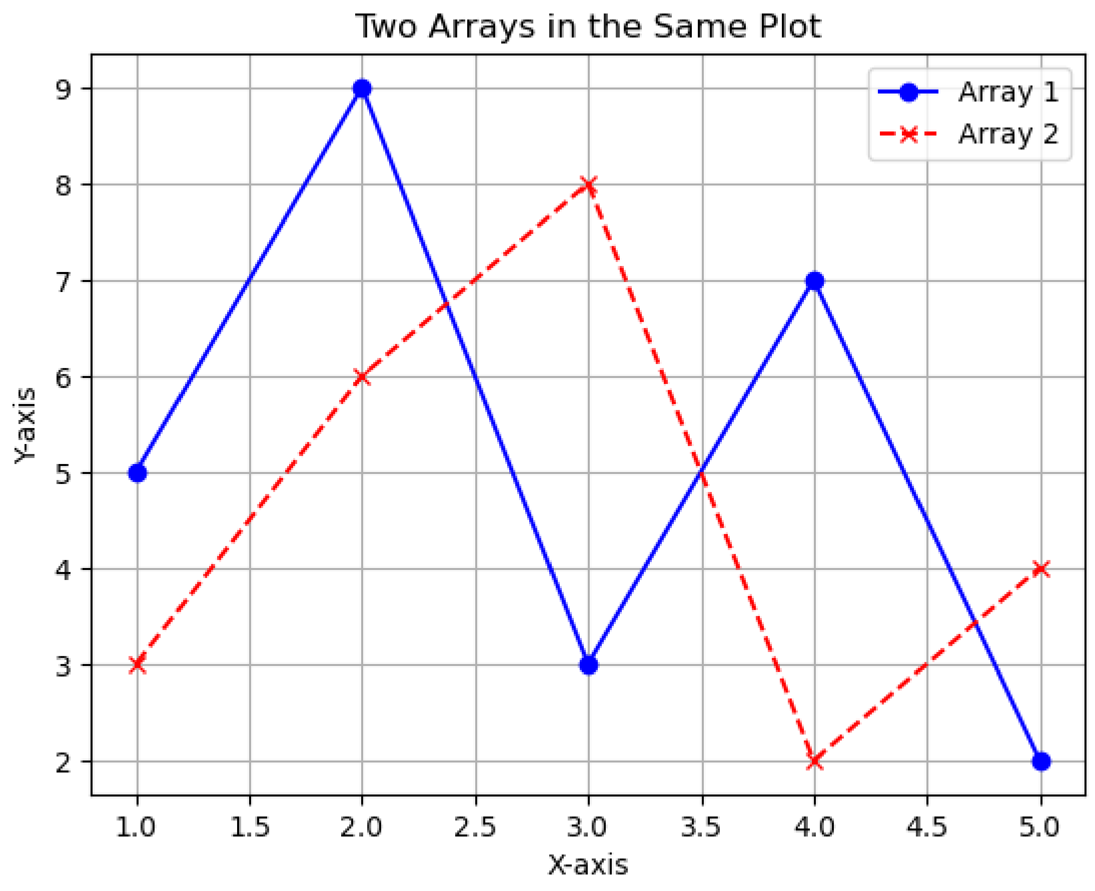
```
[ 1  4  9 16 25]
3.0
```

In [133]:

```
np.sum?
```

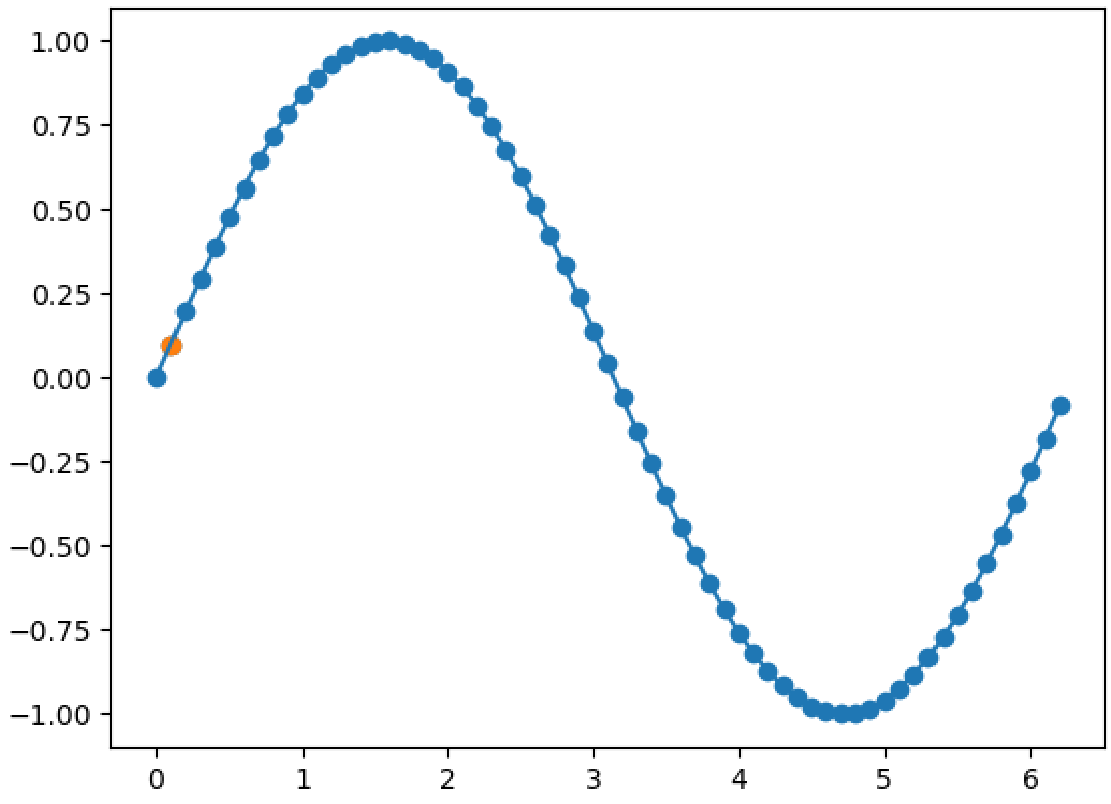
In [160]:

```python
import matplotlib.pyplot as plt

# Sample data for two arrays
x = np.array([1, 2, 3, 4, 5])
y1 = np.array([5, 9, 3, 7, 2])
y2 = np.array([3, 6, 8, 2, 4])

# Plot both arrays on the same plot
#plt.plot(x,y1, label='Array 1')
#plt.plot(x,y2, label='Array 2')
plt.plot(x, y1, label='Array 1', marker='o', linestyle='-', color='blue')
plt.plot(x, y2, label='Array 2', marker='x', linestyle='--', color='red')

#plt.scatter(x,y2, label='Array 1', marker='x')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Two Arrays in the Same Plot')

# Add legend to distinguish between the two arrays
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



In [159]:

```
x=np.arange(0,2*np.pi,0.1) #pi=3.14   np.arange(starting, ending, step size)
y=np.sin(x)
plt.plot(x,y)
plt.scatter(x,y)
plt.scatter(0.1,0.09983342)
plt.show()
print(x)
print(y)
```



```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7
 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5
 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3
 5.4 5.5 5.6 5.7 5.8 5.9 6.  6.1 6.2]
[ 0.          0.09983342  0.19866933  0.29552021  0.38941834  0.47942554
  0.56464247  0.64421769  0.71735609  0.78332691  0.84147098  0.89120736
  0.93203909  0.96355819  0.98544973  0.99749499  0.9995736   0.99166481
  0.97384763  0.94630009  0.90929743  0.86320937  0.8084964   0.74570521
  0.67546318  0.59847214  0.51550137  0.42737988  0.33498815  0.23924933
  0.14112001  0.04158066 -0.05837414 -0.15774569 -0.2555411  -0.35078323
 -0.44252044 -0.52983614 -0.61185789 -0.68776616 -0.7568025  -0.81827711
 -0.87157577 -0.91616594 -0.95160207 -0.97753012 -0.993691   -0.99992326
 -0.99616461 -0.98245261 -0.95892427 -0.92581468 -0.88345466 -0.83226744
 -0.77276449 -0.70554033 -0.63126664 -0.55068554 -0.46460218 -0.37387666
 -0.2794155  -0.1821625  -0.0830894 ]
```

In [146]:

```
plt.plot?
```

Example 6: Broadcasting

In [161]:

```
arr = np.array([1, 2, 3])
scalar = 2

result = arr + scalar
print(result)
```

```
[3 4 5]
```

In Python, a list is a mutable, ordered collection of elements. It is one of the most versatile and commonly used data structures in Python. Lists can contain elements of different data types, such as integers, strings, floats, or even other lists. The elements in a list are separated by commas and enclosed within square brackets '[ ]'.

In [167]:

```
# Creating lists
empty_list = []
single_element_list = [42]
multiple_elements_list = [1, 2, 3, 4, 5]
mixed_data_list = ['apple', 42, 3.14, True]
nested_list = [[1, 2], ['a', 'b']]

#print(nested_list)
#multiple_elements_list[1]  #starts counting from 0

# Accessing elements in a list
print('1st element of multiple_elements_list= ',multiple_elements_list[0])
print('3rd element of mixed_data_list= ',mixed_data_list[2])

# Modifying elements in a list
multiple_elements_list[0] = 10
print('multiple_elements_list: ',multiple_elements_list)
```

```
1st element of multiple_elements_list=  1
3rd element of mixed_data_list=  3.14
multiple_elements_list:  [10, 2, 3, 4, 5]
```

List Concatenation:

In [168]:

```
# Concatenate two lists
list1 = [1, 2, 3]
list2 = [4, 5, 6]
result = list1 + list2
print(result)
```

```
[1, 2, 3, 4, 5, 6]
```

List Slicing:

In [171]:

```
# Get a slice of a list
numbers = [1, 2, 3, 4, 5]
slice_result = numbers[1:3] #starting from 0th, ending n-1 ; if n=4, n-1=3
print(slice_result)
```

```
[2, 3]
```

String Concatenation:

In [172]:

```
# Concatenate two strings
string1 = "Hello "
string2 = "World!"
result = string1 + string2
print(result)
```

```
Hello World!
```

String Splitting:

In [173]:

```
# Split a string into a list of substrings
text = "Python is awesome"
words = text.split()
print(words)
```

```
['Python', 'is', 'awesome']
```

String Joining:

In [174]:

```
# Join a list of strings into a single string
words = ['Python', 'is', 'awesome']
text = ' '.join(words)
print(text)
```

```
Python is awesome
```

List Comprehensions:

In [175]:

```
# Generate a new list using list comprehension
numbers = [1, 2, 3, 4, 5]
squared_numbers = [num**2 for num in numbers]   #** = to the power
print(squared_numbers)
```

```
[1, 4, 9, 16, 25]
```

Filtering with List Comprehensions:

In [176]:

```
# Filter elements in a list using list comprehension
numbers = [1, 2, 3, 4, 5]
even_numbers = [num for num in numbers if num % 2 == 0]
print(even_numbers)
```

```
[2, 4]
```

In [ ]:

```
#you have a list of numbers, find out the even numbers
```

Reversing a List:

In [177]:

```
# Reverse a list
numbers = [1, 2, 3, 4, 5]
numbers.reverse()
print(numbers)
```

```
[5, 4, 3, 2, 1]
```

Finding Maximum and Minimum:

In [178]:

```
# Find the maximum and minimum value in a list
numbers = [10, 5, 20, 15, 25]
max_value = max(numbers)
min_value = min(numbers)
print(max_value)
print(min_value)
```

```
25
5
```

In [180]:

```
numbers = np.array([1,2,3,8,10,3,1,0])
m = np.max(numbers)
m
```

Out[180]:

```
10
```

In [182]:

```
np.linspace(1,10,101)
```

Out[182]:

```
array([ 1.  ,  1.09,  1.18,  1.27,  1.36,  1.45,  1.54,  1.63,  1.72,
        1.81,  1.9 ,  1.99,  2.08,  2.17,  2.26,  2.35,  2.44,  2.53,
        2.62,  2.71,  2.8 ,  2.89,  2.98,  3.07,  3.16,  3.25,  3.34,
        3.43,  3.52,  3.61,  3.7 ,  3.79,  3.88,  3.97,  4.06,  4.15,
        4.24,  4.33,  4.42,  4.51,  4.6 ,  4.69,  4.78,  4.87,  4.96,
        5.05,  5.14,  5.23,  5.32,  5.41,  5.5 ,  5.59,  5.68,  5.77,
        5.86,  5.95,  6.04,  6.13,  6.22,  6.31,  6.4 ,  6.49,  6.58,
        6.67,  6.76,  6.85,  6.94,  7.03,  7.12,  7.21,  7.3 ,  7.39,
        7.48,  7.57,  7.66,  7.75,  7.84,  7.93,  8.02,  8.11,  8.2 ,
        8.29,  8.38,  8.47,  8.56,  8.65,  8.74,  8.83,  8.92,  9.01,
        9.1 ,  9.19,  9.28,  9.37,  9.46,  9.55,  9.64,  9.73,  9.82,
        9.91, 10.  ])
```

List Sorting with Custom Key Function:

In [183]:

```
# Sort a list of strings based on the length of each string
words = ['apple', 'banana', 'orange', 'grape', 'kiwi']
sorted_words = sorted(words, key=len)
print(sorted_words)
```

```
['kiwi', 'apple', 'grape', 'banana', 'orange']
```

In [ ]:

```

```

List Intersection and Union:

In [186]:

```
# Find the intersection and union of two lists
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]

intersection = list(set(list1) & set(list2))
union = list(set(list1) | set(list2))

print(intersection)
print(union)
```

```
[4, 5]
[1, 2, 3, 4, 5, 6, 7, 8]
```

String Palindrome Check:

In [187]:

```
# Check if a string is a palindrome (reads the same backward as forward)
def palindrome(s):
    return s == s[::-1]

word1 = "radar"
word2 = "python"

print(palindrome(word1))
print(palindrome(word2))
```

```
True
False
```

List Chunking:

In [188]:

```
# Chunk a list into smaller sublists of a given size
def chunk_list(lst, chunk_size):
    return [lst[i:i+chunk_size] for i in range(0, len(lst), chunk_size)]

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
chunked_numbers = chunk_list(numbers, 3)
print(chunked_numbers)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10]]
```

Counting Character Occurrences:

In [189]:

```
# Count the occurrences of each character in a string
text = "Hello, Python!"

char_count = {}
for char in text:
    char_count[char] = char_count.get(char, 0) + 1

print(char_count)
```

```
{'H': 1, 'e': 1, 'l': 2, 'o': 2, ',': 1, ' ': 1, 'P': 1, 'y': 1, 't': 1,
'h': 1, 'n': 1, '!': 1}
```

Flattening Nested Lists:

In [190]:

```
# Flatten a nested list into a single list
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

flattened_list = [item for sublist in nested_list for item in sublist]
print(flattened_list)
```

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

Time Indexing

In [192]:

```
import pandas as pd

dates = pd.date_range(start='2023-01-01', periods=5, freq='D')
data = [10, 20, 30, 40, 50]
time_series = pd.Series(data, index=dates)
print(time_series)
```

```
2023-01-01    10
2023-01-02    20
2023-01-03    30
2023-01-04    40
2023-01-05    50
Freq: D, dtype: int64
```

Resampling and Aggregating

In [193]:

```
monthly_data = time_series.resample('M').mean()
print(monthly_data)
```

```
2023-01-31    30.0
Freq: M, dtype: float64
```

In Python, a tuple is an immutable, ordered collection of elements. It is similar to a list, but unlike lists, tuples cannot be modified after creation, which is why they are referred to as "immutable." Once a tuple is created, you cannot add, remove, or change elements in it.

Tuples are defined using parentheses '()' and can contain elements of different data types, such as integers, strings, floats, or even other tuples. The elements in a tuple are separated by commas.

In [194]:

```
# Creating tuples
empty_tuple = ()
single_element_tuple = (42,)  # Note the trailing comma for single-element tuples
multiple_elements_tuple = (1, 2, 3, 4, 5)
mixed_data_tuple = ('apple', 42, 3.14, True)
nested_tuple = ((1, 2), ('a', 'b'))

# Accessing elements in a tuple
print('1st element of multiple_elements_tuple= ',multiple_elements_tuple[0])
print('3rd element of mixed_data_tuple= ',mixed_data_tuple[2])

# Tuple unpacking
x, y, z = (10, 20, 30)
print('x= ',x)
print('y= ',y)
print('z= ',z)
```

```
1st element of multiple_elements_tuple=  1
3rd element of mixed_data_tuple=  3.14
x=  10
y=  20
z=  30
```

In [ ]:

```

```

# Assigment: Week 2

Question 1

In [196]:

```
#b = np.arange(1,7)
#print('b= ',b)

c = np.arange(1,7).reshape(2,3)
print(c)

d = np.array([[1,2,3],[4,5,6]])
print(np.add(c,d))
```

```
[[1 2 3]
 [4 5 6]]
[[ 2  4  6]
 [ 8 10 12]]
```

Question 2

In [197]:

```
t1 = (1,2,"tuple",4)
t2 = (5,6,7)
```

In [198]:

```
t1.append(5)
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_14152\1055591270.py in <module>
----> 1 t1.append(5)

AttributeError: 'tuple' object has no attribute 'append'
```

In [200]:

```
x = t2[t1[1]]
```

In [201]:

```
t3 = t1 + t2
```

In [202]:

```
t3 = (t1,t2)
```

In [203]:

```
t3 = (list(t1),list(t2))
```

Question 3

In [204]:

```
d1 = {1: 'Python', 2:[1,2,3]}
d1
```

Out[204]:

```
{1: 'Python', 2: [1, 2, 3]}
```

In [205]:

```
d1[2].append(4)
d1
```

Out[205]:

```
{1: 'Python', 2: [1, 2, 3, 4]}
```

In [207]:

```
x = d1[0]
x
```

```
---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_14152\3149442198.py in <module>
----> 1 x = d1[0]
      2 x

KeyError: 0
```

In [208]:

```
d1["one"]=1
d1
```

Out[208]:

```
{1: 'Python', 2: [1, 2, 3, 4], 'one': 1}
```

In [209]:

```
d1.update({"one":2})
```

In [210]:

```
d1
```

Out[210]:

```
{1: 'Python', 2: [1, 2, 3, 4], 'one': 2}
```

```
Question 5
```

In [214]:

```
S1 = {1,2,3}
S2 = {5,6,3}
S1.add(4)
S2.add("4")
S1- S2
```

Out[214]:

```
{1, 2, 4}
```

In [215]:

```
S1 = "Hello"
S2 = "World"
```

```
In [216]:
```

```
S1 + "" + S2
```

```
Out[216]:

'HelloWorld'
```

```
In [217]:
```

```
S1[0:]+""+S2[0:]
```

```
Out[217]:

'HelloWorld'
```

```
In [218]:
```

```
"{}{}".format(S1,S2)
```

```
Out[218]:

'HelloWorld'
```

```
In [220]:
```

```
S1[:-1]+""+S2[S2:-1]
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_14152\902785976.py in <module>
----> 1 S1[:-1]+""+S2[S2:-1]

TypeError: slice indices must be integers or None or have an __index__ met
hod
```

Question 6

```
In [221]:
```

```
arr = np.array([[1, 9, 10], [3, 7, 6], [12, 8, 0]])
```

```
In [224]:
```

```
arr[1: 2]
```

```
Out[224]:

array([[3, 7, 6]])
```

In [225]:

```
np.sum(arr, axis = 0)
```

Out[225]:

```
array([16, 24, 16])
```

In [227]:

```
np.sum(arr, axis = 1)
```

Out[227]:

```
array([20, 16, 20])
```

In [228]:

```
np.sum([[1, 9, 10], [3, 7, 6], [12, 8, 0]])
```

Out[228]:

```
56
```

Question 7

In [229]:

```
mat = np.matrix(“5, 9, 10; 2, 5, 4; 1, 9, 8; 2, 6, 8”)
mat1 = np.matrix(“1, 2, 3, 4”)
mat2 = np.insert(mat, 1, mat1, axis= 1)
print(mat2)
```

```
  File "C:\Users\Nilakshi\AppData\Local\Temp\ipykernel_14152\1495859806.p
y", line 1
    mat = np.matrix(“5, 9, 10; 2, 5, 4; 1, 9, 8; 2, 6, 8”)
                    ^
SyntaxError: invalid character '“' (U+201C)
```

Question 9

In [230]:

```
c = np.arange(start = 1, stop = 20, step = 3)
```

In [231]:

```
c[5]
```

Out[231]:

```
16
```

# Problem 1

```
Question: Simulating Sales Data

You are working for a retail company, and your task is to simulate sales data for
different products. You have a list of products, their base prices, and the number of
sales for each product. Your goal is to calculate the total revenue and profit for the
company.
```

In [67]:

```python
import numpy as np
import pandas as pd

# List of products and their base prices
products = ['Product A', 'Product B', 'Product C', 'Product D']
base_prices = [10, 15, 20, 25] #in dollars

# Simulate the number of sales for each product (random integers between 50 and 100)
np.random.seed(42)  # Set a random seed for reproducibility
num_sales = np.random.randint(50, 101, size=len(products))

# Calculate the total revenue and profit
total_revenue = sum(base_prices[i] * num_sales[i] for i in range(len(products)))
total_cost = sum(5 * num_sales[i] for i in range(len(products)))  # Assuming a constant
total_profit = total_revenue - total_cost

# Create a DataFrame to store the sales data
sales_data = pd.DataFrame({
    'Product': products,
    'Base Price': base_prices,
    'Number of Sales': num_sales,
})

# Calculate the revenue and profit for each product
sales_data['Revenue'] = sales_data['Base Price'] * sales_data['Number of Sales']
sales_data['Cost'] = 5 * sales_data['Number of Sales']
sales_data['Profit'] = sales_data['Revenue'] - sales_data['Cost']

print("Sales Data:")
print(sales_data)
print("\nTotal Revenue: $", total_revenue)
print("Total Profit: $", total_profit)
```

```
Sales Data:
     Product  Base Price  Number of Sales  Revenue  Cost  Profit
0  Product A          10               88      880   440     440
1  Product B          15               78     1170   390     780
2  Product C          20               64     1280   320     960
3  Product D          25               92     2300   460    1840

Total Revenue: $ 5630
Total Profit: $ 4020
```

# Problem 2

```
Question: Analyzing Student Grades
```

```
Suppose you are given a CSV file containing information about student grades in
different subjects. The CSV file has the following columns: "StudentID", "Name",
"Subject", and "Grade". Each row represents a student's grade in a specific subject.

Your task is to:

Load the data from the CSV file into a pandas DataFrame.
Calculate the average grade for each subject.
Find the student with the highest average grade.
Determine the number of students who failed (grade less than 60) in each subject.
Concatenate the student's name and subject to form a new column "Student_Subject" in
the DataFrame.
Convert the DataFrame into a NumPy array and perform element-wise arithmetic
operations.
Use string operations to extract the first name and last name of the student.
```