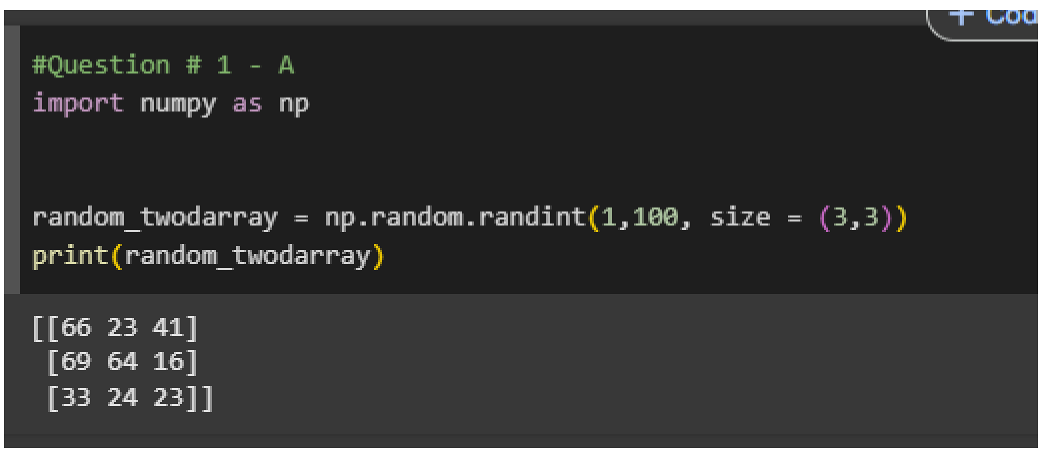
Question no: 1 (NumPy):

A- Using NumPy, create a 2D array (3 rows, 3 columns) filled with random integers between 1 and 100.

Solution



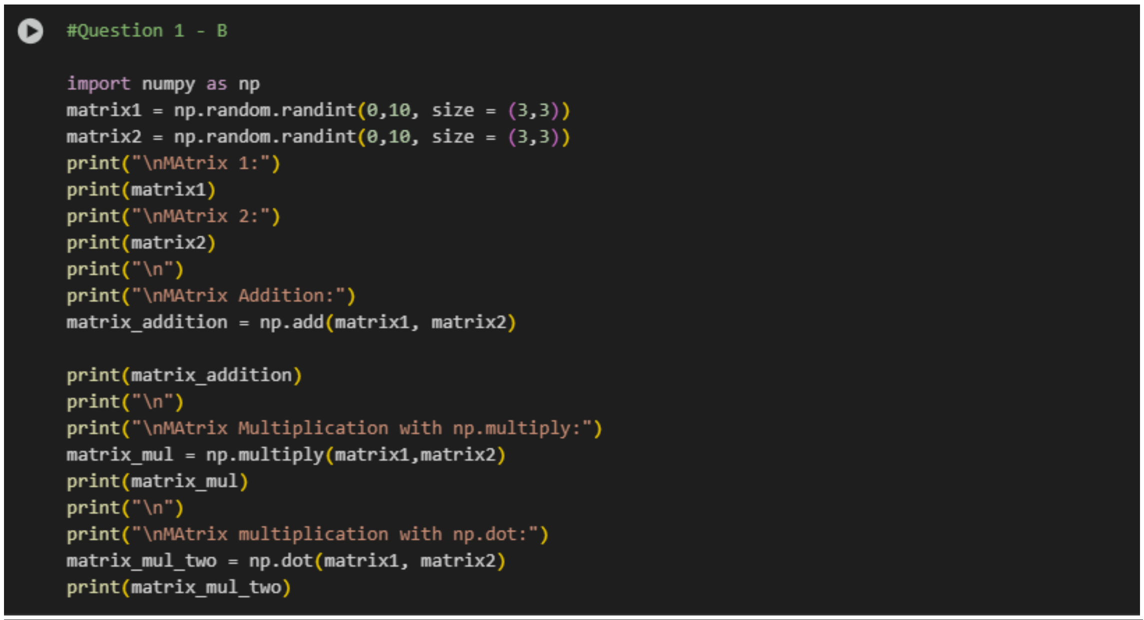
```python
#Question # 1 - A
import numpy as np


random_twodarray = np.random.randint(1,100, size = (3,3))
print(random_twodarray)
```

```
[[66 23 41]
 [69 64 16]
 [33 24 23]]
```

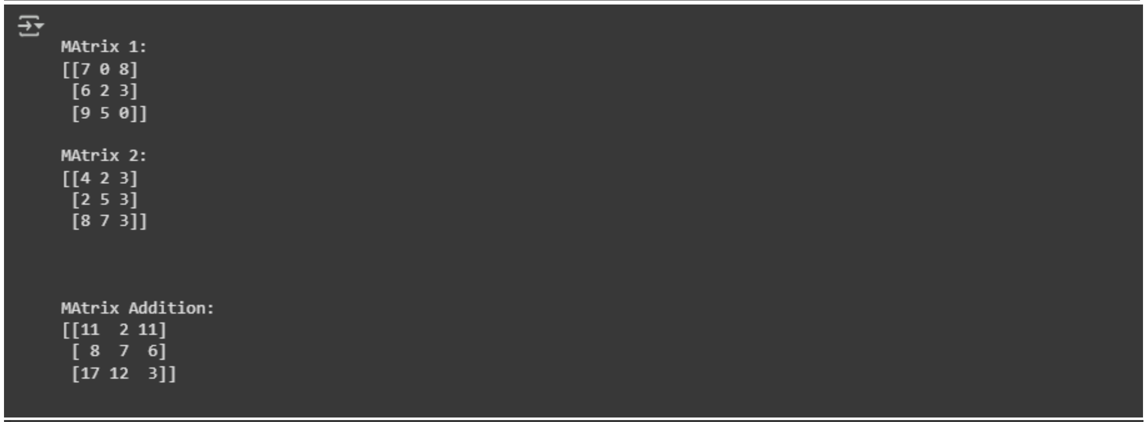
B- **Using NumPy, create two random integer matrices of shape (3x3). Perform:**

- Matrix addition
- Element-wise multiplication
- Matrix multiplication (np.dot)

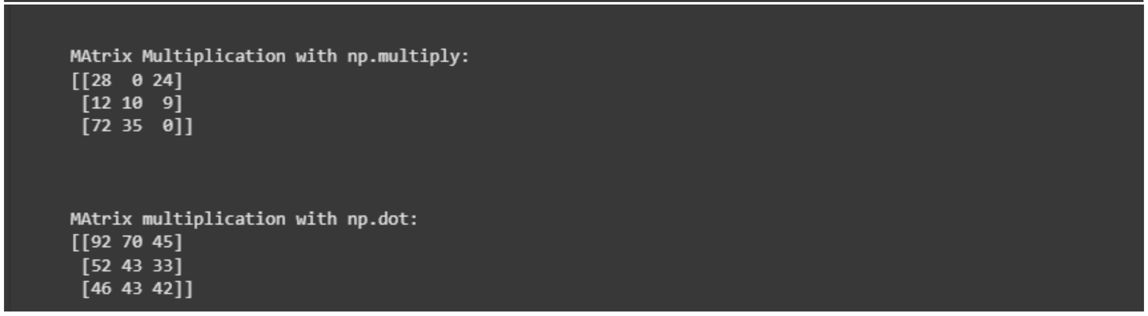
Solution:

- 

```
#Question 1 - B

import numpy as np
matrix1 = np.random.randint(0,10, size = (3,3))
matrix2 = np.random.randint(0,10, size = (3,3))
print("\nMAtrix 1:")
print(matrix1)
print("\nMAtrix 2:")
print(matrix2)
print("\n")
print("\nMAtrix Addition:")
matrix_addition = np.add(matrix1, matrix2)

print(matrix_addition)
print("\n")
print("\nMAtrix Multiplication with np.multiply:")
matrix_mul = np.multiply(matrix1,matrix2)
print(matrix_mul)
print("\n")
print("\nMAtrix multiplication with np.dot:")
matrix_mul_two = np.dot(matrix1, matrix2)
print(matrix_mul_two)
```



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

MAtrix 2:
[[4 2 3]
 [2 5 3]
 [8 7 3]]



MAtrix Addition:
[[11  2 11]
 [ 8  7  6]
 [17 12  3]]
```

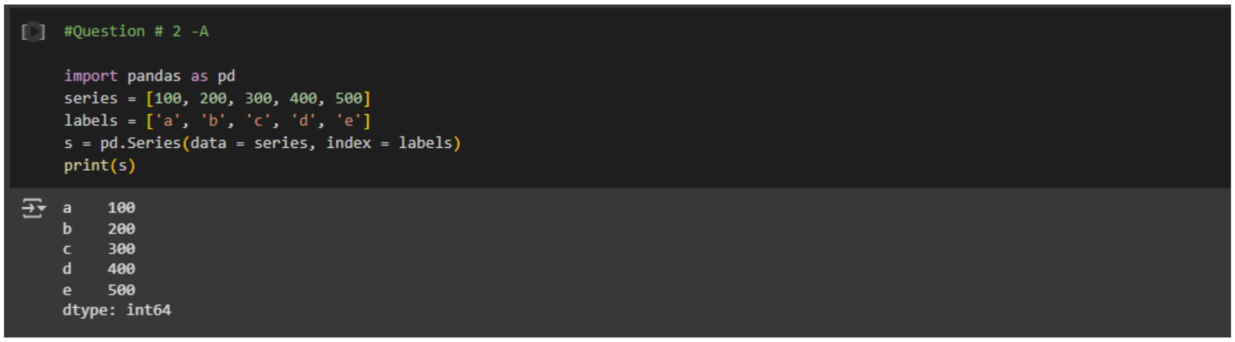


```
MAtrix Multiplication with np.multiply:
[[28  0 24]
 [12 10  9]
 [72 35  0]]



MAtrix multiplication with np.dot:
[[92 70 45]
 [52 43 33]
 [46 43 42]]
```

Question no: 2 (pandas):

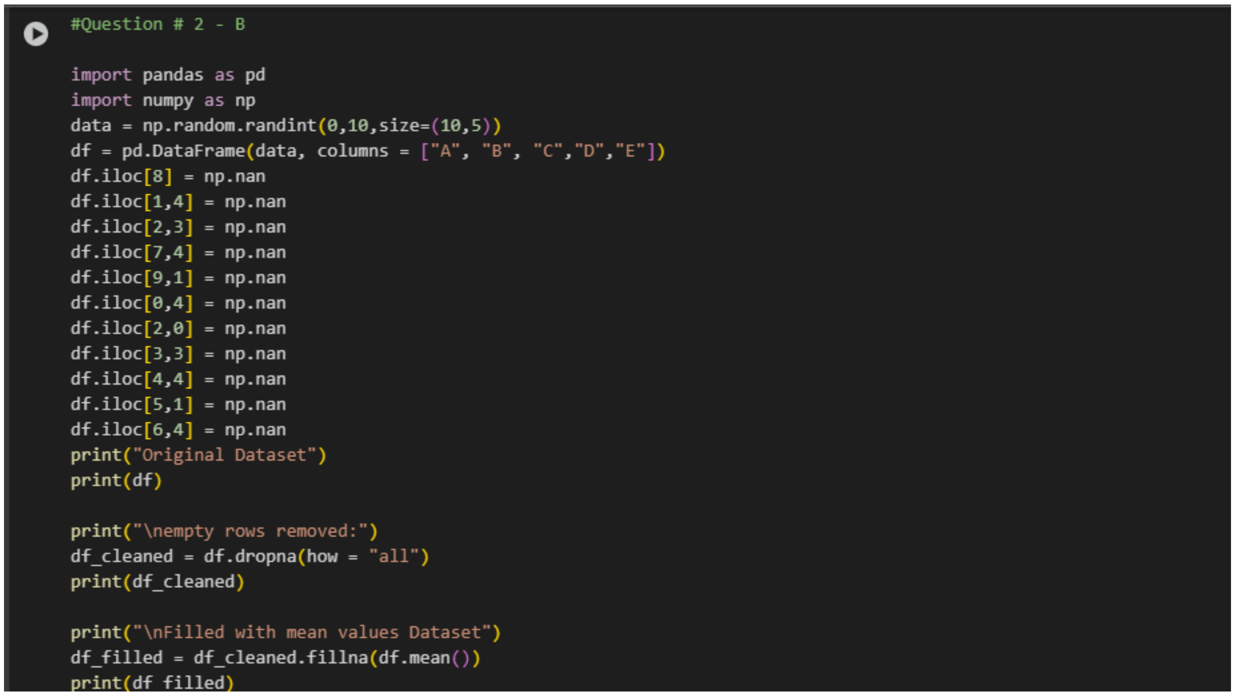
A- Using Pandas, create a Series with custom index labels ['a','b','c','d','e'] for the list [100, 200, 300, 400, 500].

Solution:



```
#Question # 2 -A

import pandas as pd
series = [100, 200, 300, 400, 500]
labels = ['a', 'b', 'c', 'd', 'e']
s = pd.Series(data = series, index = labels)
print(s)
```

```
a    100
b    200
c    300
d    400
e    500
dtype: int64
```



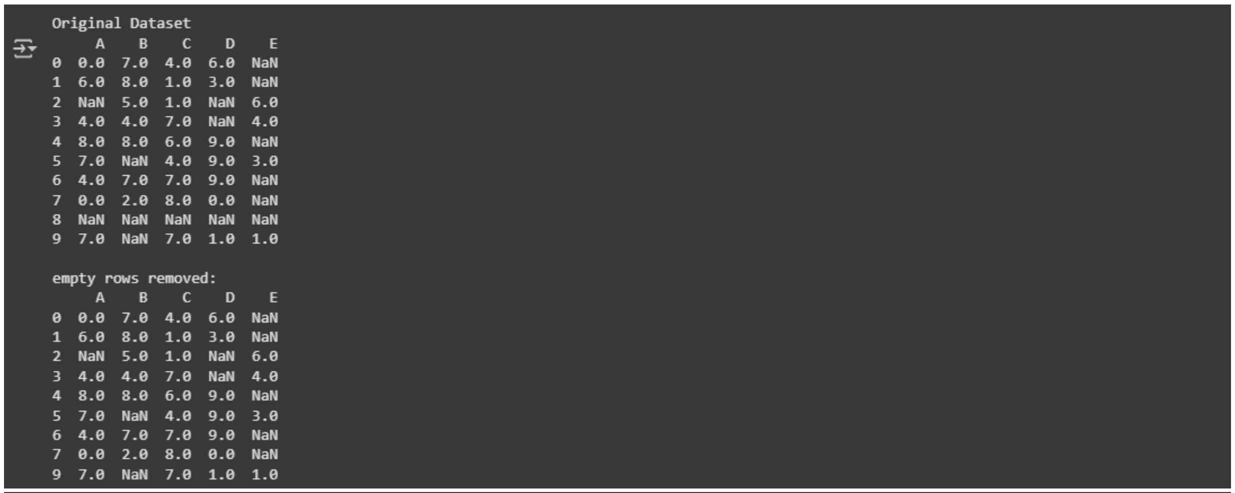
```
#Question # 2 - B

import pandas as pd
import numpy as np
data = np.random.randint(0,10,size=(10,5))
df = pd.DataFrame(data, columns = ["A", "B", "C","D","E"])
df.iloc[8] = np.nan
df.iloc[1,4] = np.nan
df.iloc[2,3] = np.nan
df.iloc[7,4] = np.nan
df.iloc[9,1] = np.nan
df.iloc[0,4] = np.nan
df.iloc[2,0] = np.nan
df.iloc[3,3] = np.nan
df.iloc[4,4] = np.nan
df.iloc[5,1] = np.nan
df.iloc[6,4] = np.nan
print("Original Dataset")
print(df)

print("\nempty rows removed:")
df_cleaned = df.dropna(how = "all")
print(df_cleaned)

print("\nFilled with mean values Dataset")
df_filled = df_cleaned.fillna(df.mean())
print(df_filled)
```

B- **Using Pandas, create a DataFrame with some missing values (NaN).**

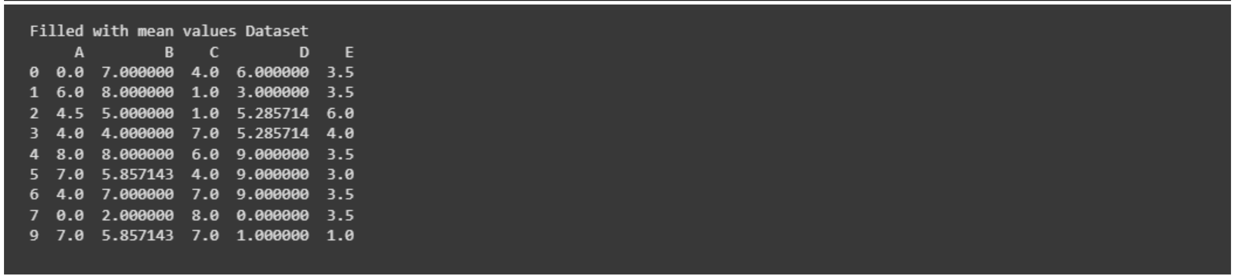
- Fill missing values with the column mean

Drop rows where all values are missing

<u>Solution:</u>



```
Original Dataset
     A    B    C    D    E
0  0.0  7.0  4.0  6.0  NaN
1  6.0  8.0  1.0  3.0  NaN
2  NaN  5.0  1.0  NaN  6.0
3  4.0  4.0  7.0  NaN  4.0
4  8.0  8.0  6.0  9.0  NaN
5  7.0  NaN  4.0  9.0  3.0
6  4.0  7.0  7.0  9.0  NaN
7  0.0  2.0  8.0  0.0  NaN
8  NaN  NaN  NaN  NaN  NaN
9  7.0  NaN  7.0  1.0  1.0

empty rows removed:
     A    B    C    D    E
0  0.0  7.0  4.0  6.0  NaN
1  6.0  8.0  1.0  3.0  NaN
2  NaN  5.0  1.0  NaN  6.0
3  4.0  4.0  7.0  NaN  4.0
4  8.0  8.0  6.0  9.0  NaN
5  7.0  NaN  4.0  9.0  3.0
6  4.0  7.0  7.0  9.0  NaN
7  0.0  2.0  8.0  0.0  NaN
9  7.0  NaN  7.0  1.0  1.0
```



```
Filled with mean values Dataset
     A         B    C         D    E
0  0.0  7.000000  4.0  6.000000  3.5
1  6.0  8.000000  1.0  3.000000  3.5
2  4.5  5.000000  1.0  5.285714  6.0
3  4.0  4.000000  7.0  5.285714  4.0
4  8.0  8.000000  6.0  9.000000  3.5
5  7.0  5.857143  4.0  9.000000  3.0
6  4.0  7.000000  7.0  9.000000  3.5
7  0.0  2.000000  8.0  0.000000  3.5
9  7.0  5.857143  7.0  1.000000  1.0
```