Python Programming

Lab:- 18(Numpy Function)

Student Id:- AF0417098

Student Name:- Ankush

- **NumPy is a powerful library in Python for numerical computing, especially useful for working with arrays and performing operations on them efficiently.**

- **Here's a detailed guide on the most important NumPy functions:**

1. Array Creation Functions

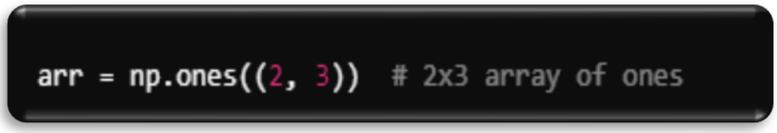
- np.array(): Creates a NumPy array from a Python list or tuple.

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

- np.zeros(): Creates an array filled with zeros.

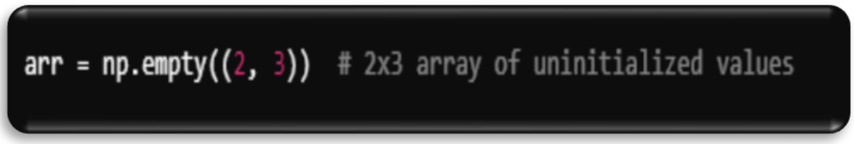
```
arr = np.zeros((3, 4))  # 3x4 array of zeros
```

- np.ones(): Creates an array filled with ones.



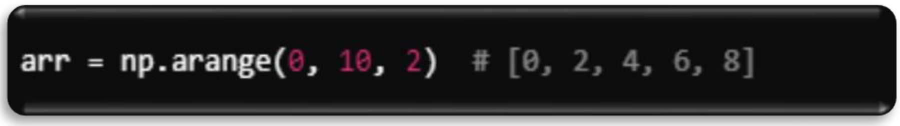
```
arr = np.ones((2, 3))  # 2x3 array of ones
```

- np.empty(): Creates an uninitialized array of given shape.



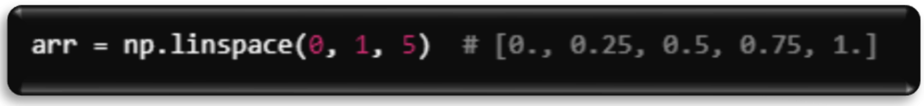
```python
arr = np.empty((2, 3))  # 2x3 array of uninitialized values
```

- **np.arange():** **Returns evenly spaced values within a given interval.**



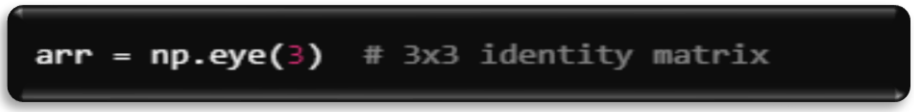
```python
arr = np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
```

- **np.linspace():** **Returns evenly spaced numbers over a specified interval.**



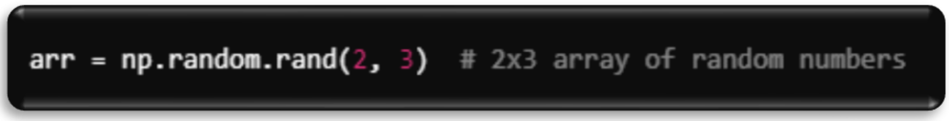
```python
arr = np.linspace(0, 1, 5)  # [0., 0.25, 0.5, 0.75, 1.]
```

- **np.eye():** **Creates a 2D identity matrix.**



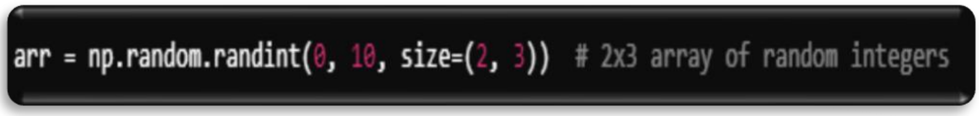
```python
arr = np.eye(3)  # 3x3 identity matrix
```

- **np.random.rand():** **Generates an array of random numbers between 0 and 1.**



```python
arr = np.random.rand(2, 3)  # 2x3 array of random numbers
```

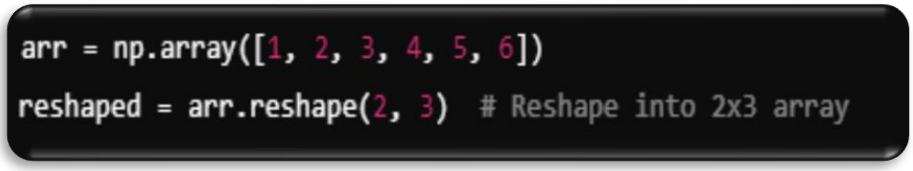
- **np.random.randint():** **Generates random integers between specified values.**



```python
arr = np.random.randint(0, 10, size=(2, 3))  # 2x3 array of random integers
```

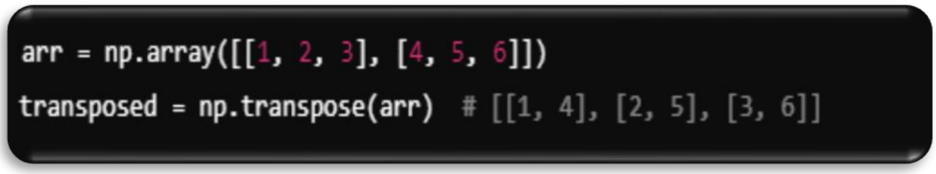
2. Array Manipulation Functions:-

- **np.reshape():** Reshapes an array without changing its data.



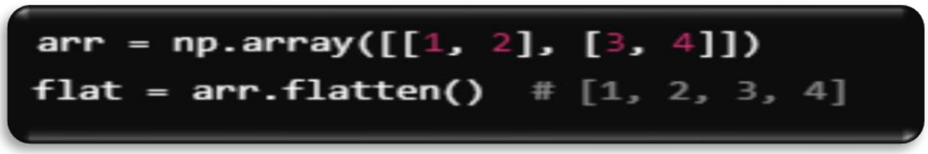
```python
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped = arr.reshape(2, 3)  # Reshape into 2x3 array
```

- **np.transpose():** Transposes the array (flips axes).



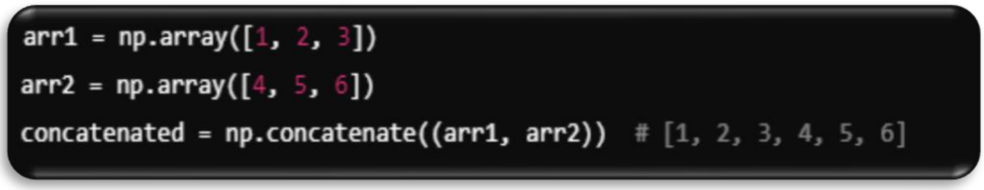
```python
arr = np.array([[1, 2, 3], [4, 5, 6]])
transposed = np.transpose(arr)  # [[1, 4], [2, 5], [3, 6]]
```

- **np.flatten():** Flattens a multi-dimensional array into a 1D array.



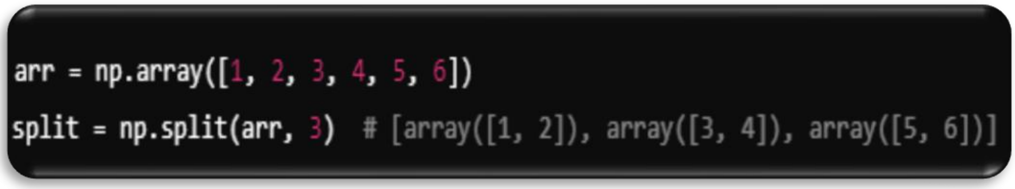
```python
arr = np.array([[1, 2], [3, 4]])
flat = arr.flatten()  # [1, 2, 3, 4]
```

- **np.concatenate():** Joins two or more arrays along an axis.



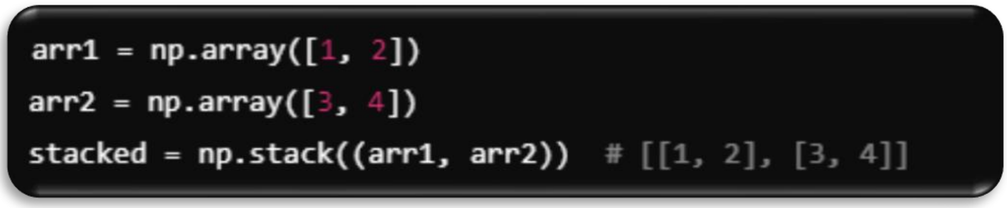
```python
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
concatenated = np.concatenate((arr1, arr2))  # [1, 2, 3, 4, 5, 6]
```

- **np.split():** Splits an array into multiple sub-arrays.



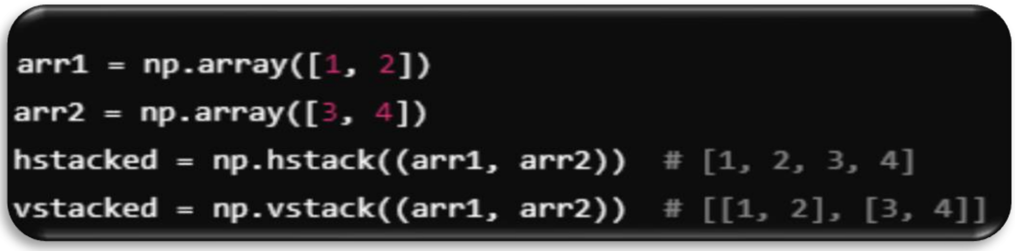
```python
arr = np.array([1, 2, 3, 4, 5, 6])
split = np.split(arr, 3)  # [array([1, 2]), array([3, 4]), array([5, 6])]
```

- **np.stack():** Stacks arrays along a new axis.



```python
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])
stacked = np.stack((arr1, arr2))  # [[1, 2], [3, 4]]
```

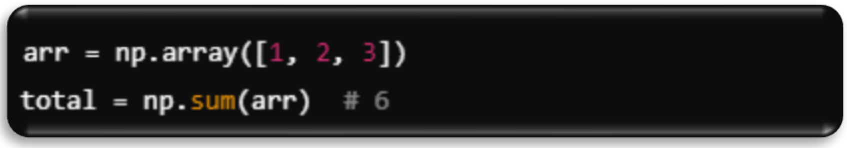
- **np.hstack() / np.vstack():** Horizontal/vertical stacking of arrays.



```python
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])
hstacked = np.hstack((arr1, arr2))  # [1, 2, 3, 4]
vstacked = np.vstack((arr1, arr2))  # [[1, 2], [3, 4]]
```

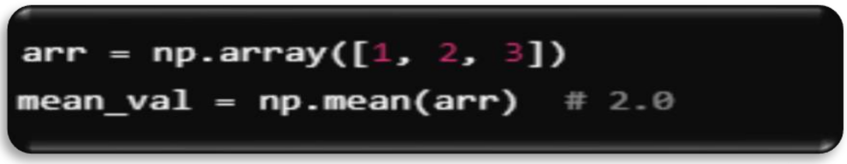
3. Mathematical Functions:-

- np.sum(): Returns the sum of array elements.



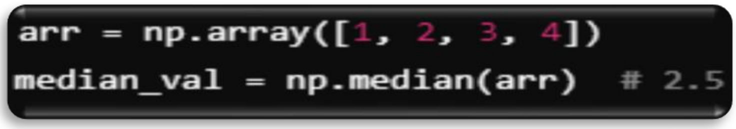
```
arr = np.array([1, 2, 3])
total = np.sum(arr)  # 6
```

- np.mean(): Computes the arithmetic mean.



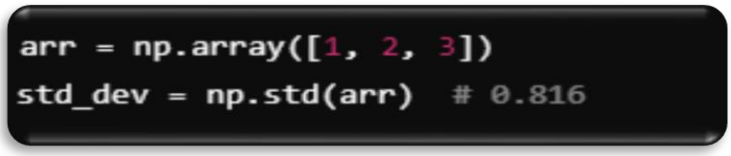
```
arr = np.array([1, 2, 3])
mean_val = np.mean(arr)  # 2.0
```

- np.median(): Returns the median of the array.



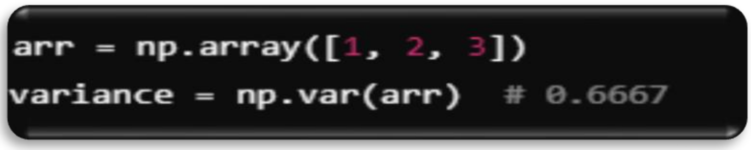
```
arr = np.array([1, 2, 3, 4])
median_val = np.median(arr)  # 2.5
```

- np.std(): Computes the standard deviation.



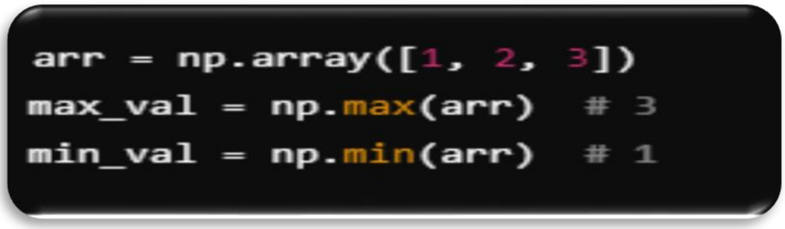
```
arr = np.array([1, 2, 3])
std_dev = np.std(arr)  # 0.816
```

- np.var(): Computes the variance.



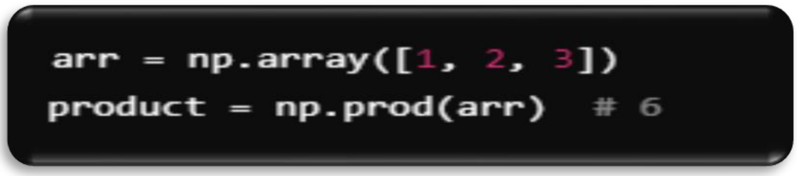
```
arr = np.array([1, 2, 3])
variance = np.var(arr)  # 0.6667
```

- np.max() / np.min(): Returns the maximum/minimum value.



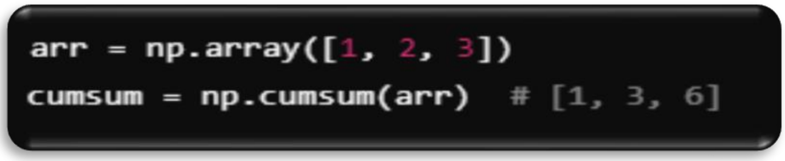
```
arr = np.array([1, 2, 3])
max_val = np.max(arr)  # 3
min_val = np.min(arr)  # 1
```

- np.prod(): Returns the product of array elements.



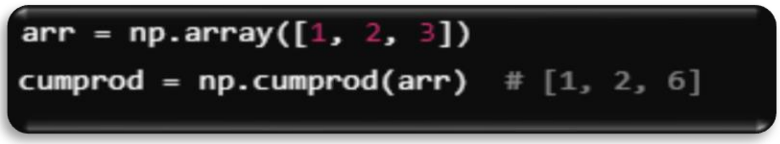
```
arr = np.array([1, 2, 3])
product = np.prod(arr)  # 6
```

- np.cumsum(): Returns the cumulative sum of the elements.



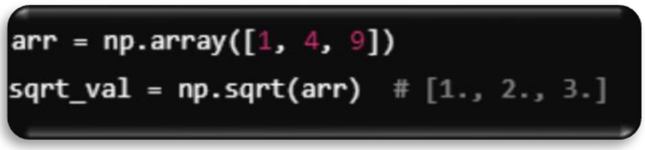
```
arr = np.array([1, 2, 3])
cumsum = np.cumsum(arr)  # [1, 3, 6]
```

- np.cumprod(): Returns the cumulative product of the elements.



```
arr = np.array([1, 2, 3])
cumprod = np.cumprod(arr)  # [1, 2, 6]
```

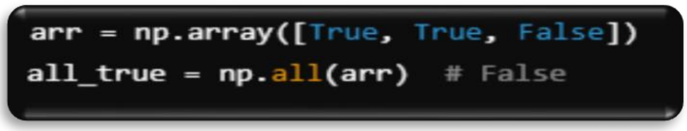
- np.sqrt(): Returns the square root of each element.



```
arr = np.array([1, 4, 9])
sqrt_val = np.sqrt(arr)  # [1., 2., 3.]
```

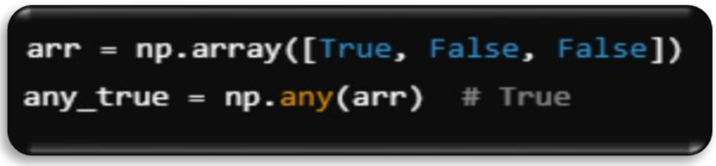
4. Logical Operations:-

- np.all(): Returns True if all elements evaluate to True.



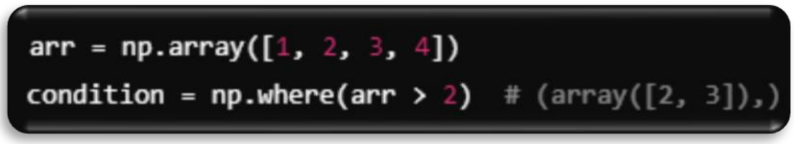
```
arr = np.array([True, True, False])
all_true = np.all(arr)  # False
```

- np.any(): Returns True if any element evaluates to True.



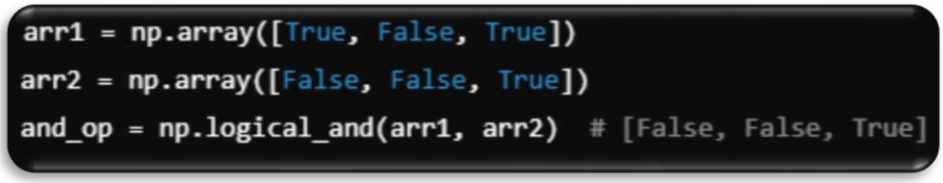
```
arr = np.array([True, False, False])
any_true = np.any(arr)  # True
```

- np.where(): Returns indices where a condition is true.



```
arr = np.array([1, 2, 3, 4])
condition = np.where(arr > 2)  # (array([2, 3]),)
```

- np.logical_and(), np.logical_or(), np.logical_not(): Logical operations.



```
arr1 = np.array([True, False, True])
arr2 = np.array([False, False, True])
and_op = np.logical_and(arr1, arr2)  # [False, False, True]
```

Assignment Questions:-

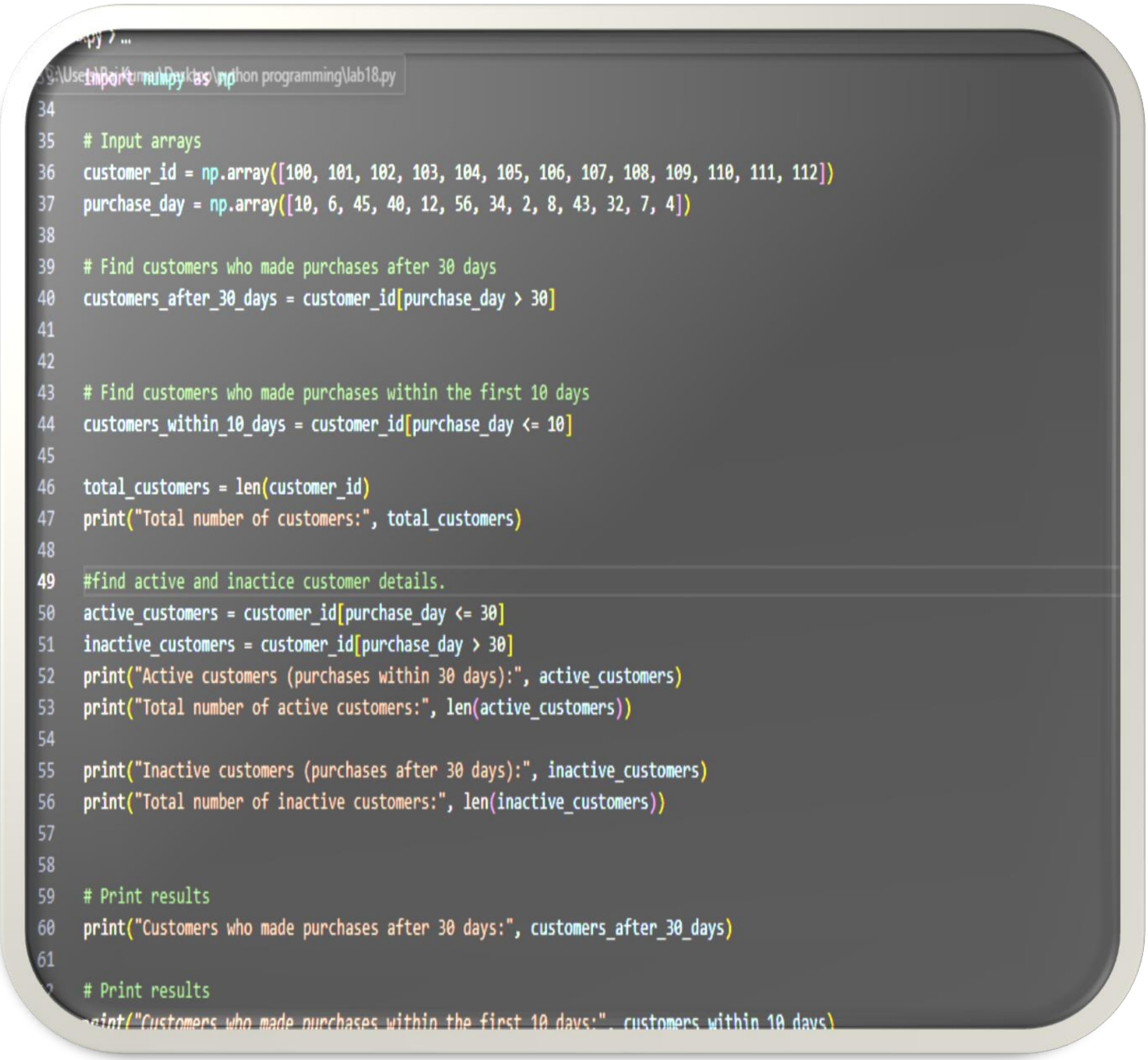
Ques1:- customer_id=np.array([100,101,102,103,104,105,106,107,108,109,110,111,112])

purchase_day=np.array([10,6,45,40,12,56,34,2,8,43,32,7,4]).

Find customer who made purchase after 30 days.

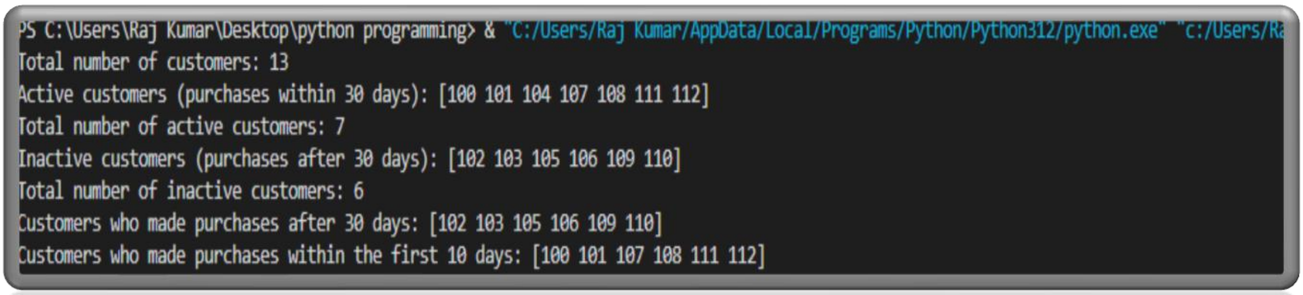
And customer who made purchase within first 10 days.

Program:-



```python
import numpy as np

# Input arrays
customer_id = np.array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112])
purchase_day = np.array([10, 6, 45, 40, 12, 56, 34, 2, 8, 43, 32, 7, 4])

# Find customers who made purchases after 30 days
customers_after_30_days = customer_id[purchase_day > 30]


# Find customers who made purchases within the first 10 days
customers_within_10_days = customer_id[purchase_day <= 10]

total_customers = len(customer_id)
print("Total number of customers:", total_customers)

#find active and inactice customer details.
active_customers = customer_id[purchase_day <= 30]
inactive_customers = customer_id[purchase_day > 30]
print("Active customers (purchases within 30 days):", active_customers)
print("Total number of active customers:", len(active_customers))

print("Inactive customers (purchases after 30 days):", inactive_customers)
print("Total number of inactive customers:", len(inactive_customers))


# Print results
print("Customers who made purchases after 30 days:", customers_after_30_days)

# Print results
print("Customers who made purchases within the first 10 days:", customers_within_10_days)
```

Output:-



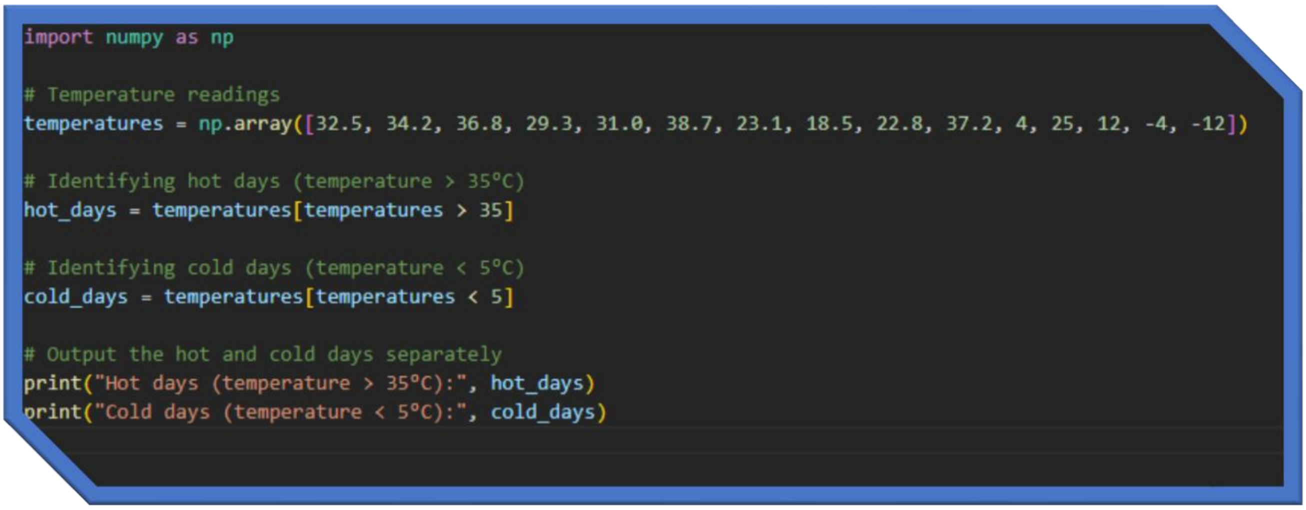
```
PS C:\Users\Raj Kumar\Desktop\python programming> & "C:/Users/Raj Kumar/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/Ra
Total number of customers: 13
Active customers (purchases within 30 days): [100 101 104 107 108 111 112]
Total number of active customers: 7
Inactive customers (purchases after 30 days): [102 103 105 106 109 110]
Total number of inactive customers: 6
Customers who made purchases after 30 days: [102 103 105 106 109 110]
Customers who made purchases within the first 10 days: [100 101 107 108 111 112]
```

Ques 2.

Suppose you have a dataset containing daily temperature readings for a city, and you want to identify days with extreme temperature conditions. Find days where the temperature either exceeded 35 degrees Celsius (hot day) or dropped below 5 degrees Celsius (cold day).

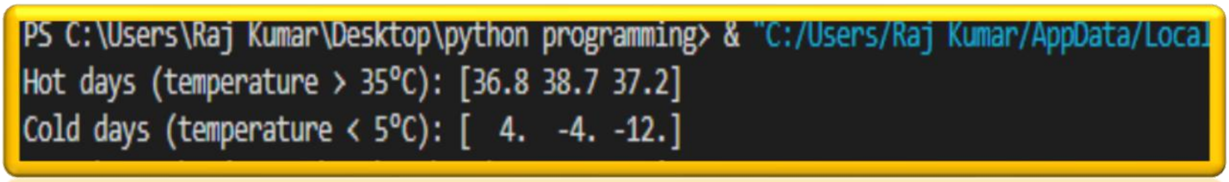
Input: temperatures = np.array([32.5, 34.2, 36.8, 29.3, 31.0, 38.7, 23.1, 18.5, 22.8, 37.2,4,25,12,-4,-12])

Program:-



```python
import numpy as np

# Temperature readings
temperatures = np.array([32.5, 34.2, 36.8, 29.3, 31.0, 38.7, 23.1, 18.5, 22.8, 37.2, 4, 25, 12, -4, -12])

# Identifying hot days (temperature > 35°C)
hot_days = temperatures[temperatures > 35]

# Identifying cold days (temperature < 5°C)
cold_days = temperatures[temperatures < 5]

# Output the hot and cold days separately
print("Hot days (temperature > 35°C):", hot_days)
print("Cold days (temperature < 5°C):", cold_days)
```

Output:-



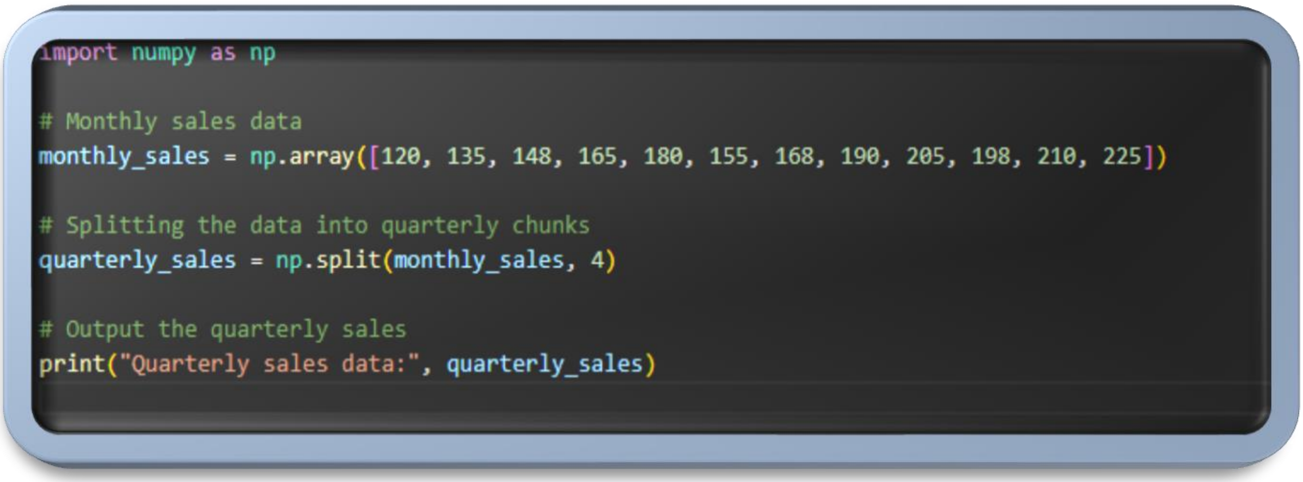
```
PS C:\Users\Raj Kumar\Desktop\python programming> & "C:/Users/Raj Kumar/AppData/Local
Hot days (temperature > 35°C): [36.8 38.7 37.2]
Cold days (temperature < 5°C): [  4.  -4. -12.]
```

Ques 3.

Suppose you have a dataset containing monthly sales data for a company, and you want to split this data into quarterly reports for analysis and reporting purposes.

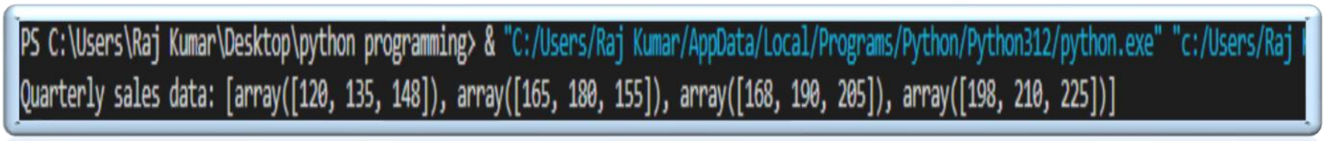
Input: monthly_sales = np.array([120, 135, 148, 165, 180, 155, 168, 190, 205, 198, 210, 225])

Program:-



```python
import numpy as np

# Monthly sales data
monthly_sales = np.array([120, 135, 148, 165, 180, 155, 168, 190, 205, 198, 210, 225])

# Splitting the data into quarterly chunks
quarterly_sales = np.split(monthly_sales, 4)

# Output the quarterly sales
print("Quarterly sales data:", quarterly_sales)
```

Output:-



```
PS C:\Users\Raj Kumar\Desktop\python programming> & "C:/Users/Raj Kumar/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/Raj
Quarterly sales data: [array([120, 135, 148]), array([165, 180, 155]), array([168, 190, 205]), array([198, 210, 225])]
```