1. Inheritance

```
class Employee:
    def __init__(self,name,id):
        self.name = name
        self.id = id
class PermanentEmployee(Employee):
    def __init__(self, name, id,salary):
        self.salary = salary
        super().__init__(name, id)
    def calculate_salary(self):
        return self.salary
class ContractEmployee(Employee):
    def __init__(self, name, id,perHour,workHour):
        self.rate = perHour
        self.hour = workHour
        super().__init__(name, id)
    def calculate_salary(self):
        return self.rate*self.hour
P_employee = PermanentEmployee("Enamul","1946",12000)
print(P_employee.calculate_salary())

e2 = ContractEmployee("Employee 2","15454",60,12)
print(e2.calculate_salary())
```

Output:

2. Polymorphism

```
class Transport:
    def __init__(self) -> None:
        pass
    def calculate_cost(self,weight,distance):
        return weight*distance

class Truck(Transport):
    def __init__(self) -> None:
        super().__init__()
    def calculate_cost(self,weight,distance):
        return weight*distance

class Ship(Transport):
```

```
    def __init__(self) -> None:
        super().__init__()
    def calculate_cost(self,weight,distance):
        return weight*2*distance

class Plane(Transport):
    def __init__(self) -> None:
        super().__init__()
    def calculate_cost(self,weight,distance):
        return weight*120*distance
print("Ship-->")
ship = Ship()
print(ship.calculate_cost(4,5))
print("Truck-->")
truck = Truck()
print(truck.calculate_cost(40,60))
print("Transport-->")
transport = Transport()
print(transport.calculate_cost(3,4))
print("Plane-->")
plane = Plane()
print(plane.calculate_cost(2,4))
```

Output

```
Ship-->

40

Truck-->

2400

Transport-->

12

Plane-->

960
```

3. Exception Handling

```
def divide_elements(values, divisor):
    result = []
    try:
        for value in values:
            r = value / divisor
            result.append(r)
    except ZeroDivisionError:
        print("Error : Division by zero is not possible")
        return
    except TypeError:
        print("Not Numeric")
        return
    return result
result = divide_elements([20,10,30,40],2)
print("Result: ",result)
```

Output

```
Result:  [10.0, 5.0, 15.0, 20.0]
```

4. Custom Exception Handling

```
class InsufficientFundsError(Exception):
    def __init__(self,message):
        super().__init__(message)

class BankAccount:
    def __init__(self,balance,min_balance):
        self.balance = balance
        self.min_balance = min_balance
    def withdraw(self,amount):
        current_balance = self.balance-amount
        if current_balance<self.min_balance:
            raise InsufficientFundsError('Insufficient Balance')
        print("Withdraw successful!")
        self.balance -= amount
try:
    account= BankAccount(100,20)
    account.withdraw(80)
    account.withdraw(80)
except InsufficientFundsError as err:
    print(err)
```

Output:

Withdraw successful!

Insufficient Balance

5. Numpy Function

```
import numpy as np
def find_avg(students):
    avg = np.mean(students,axis=1)
    mean_index = np.argmax(avg)
    highest = avg[mean_index]
    print(f"Student: {mean_index+1}\nAverage Mark: {highest}")
students = np.array([
    [80,90,23,60],
    [80, 90, 23, 40],
    [55, 30, 23, 70],
    [77, 95, 23, 60],
    [99, 20, 23, 50]
])
find_avg(students)
```

Output:

Student: 4

Average Mark: 63.75

6.Indexing and Slicing

```
import numpy as np
sales_data = np.array([
    [100,200,450,550],
    [150,250,300,700],
    [500,200,450,400],
    [350,300,200,100]
    ])
print(sales_data)
print("\nSales data for first three product : ",sales_data[:3])
print("\nSales data for all products in last two month : ",sales_data[0:4,-2:])
print("\n Sales data for a specific product and month : ",sales_data[1,3])
```

Output :

[[100 200 450 550]

[150 250 300 700]

[500 200 450 400]

[350 300 200 100]]

Sales data for first three product : [[100 200 450 550]

[150 250 300 700]

[500 200 450 400]]

Sales data for all procuts in last two month : [[450 550]

[300 700]

6. Type Casting

```
import numpy as np
data = np.array([ ["Enamul",1611,200.30], ["Mehedi",1844,100.50], ["Haq",1788,150.78],
["Rana",1756,20.50]])
print(data)
print(data.dtype)
id = data[:,1].astype(int)
print(id)
print(id.dtype)
salary = data[:,2].astype(float)
print(salary)
print(salary.dtype)
```

Output :

[[‘Enamul '1611' '200.3']

[‘Mehedi '1844' '100.5']

[‘Rasel '1788' '150.78']

[Ajay '1756' '20.5']]

<U32

[1611 1844 1788 1756]

int64

[200.3 100.5 150.78 20.5]

float64

7. Copy and View

```
import numpy as np
a = np.array([
   [1,2,3,4,5],
   [4,5,6,7,8],
   [8,9,0,1,2],
   [6,7,8,9,0]
])
print(a)
print("\nCreate a view of third row")
print(a[2,:])
print("\nCreate a copy of third column")
print(a[:,2].copy())
print("\n Modify the array")
b = a[2,:]
b[:] = 0
print(b)
print("After Modify Orginal array")
print(a)
print("\n Copy of third column and modify")
c =a[:,2].copy()
c[:] = 0
print(c)
print("After Modify Orginal Array")
print(a)
```

Output :

[[1 2 3 4 5]

[4 5 6 7 8]

[8 9 0 1 2]

[6 7 8 9 0]]

Create a view of third row

[8 9 0 1 2]

Create a copy of third column

[3 6 0 8]

Modify the array

[0 0 0 0 0]

After Modify Orginal array

[[1 2 3 4 5]

[4 5 6 7 8]

[0 0 0 0 0]

[6 7 8 9 0]]

Copy of third column and modify

[0 0 0 0]

After Modify Orginal Array

[[1 2 3 4 5]

[4 5 6 7 8]

[0 0 0 0 0]

[6 7 8 9 0]]

8. Shape and Reshape

```
import numpy as np
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
rows = 3
columns = 4
def reshape_array_pad(data, rows, columns):
    total_elements = rows * columns
    if data.size < total_elements:
        padded_data = np.pad(data, (0, total_elements - data.size), 'constant')
        print(f"Data padded with {total_elements - data.size} zeros.")
        return padded_data.reshape(rows, columns)
    else:
        return data[:total_elements].reshape(rows, columns)
reshaped_array = reshape_array_pad(data, rows, columns)
print("Reshaped 2D array:")
print(reshaped_array)
```

Output :

Data padded with 2 zeros.

Reshaped 2D array:

[[1 2 3 4]

[5 6 7 8]

[9 10 0 0]]

9. Join

```
import numpy as np
arr1 = np.array([1,2,3,4,5,6])
arr2 = np.array([6,7,8,9,1,2])
print(np.stack((arr1,arr2)))
print(np.stack((arr1,arr2),axis= 1))
```

Ouput :

[[1 2 3 4 5 6]

[6 7 8 9 1 2]]

[[1 6]

[2 7]

[3 8]

[4 9]

[5 1]

[6 2]]

10. Numpy Where

```
import numpy as np

np.where

tempeatures = np.array([17,12,15,18,19,20,30,40,42,22,38,44])

high = 40

low = 17

exceeds_temperature = np.where(tempeatures>high)[0]
```

```
print('Temperature Exceeds',exceeds_temperature)

adjusted_temperature = np.where(tempeatures<low,low,tempeatures)

print('Replaced : ',adjusted_temperature)
```

Output :

```
Temperature Exceeds [ 8 11]

Replaced :  [17 17 17 18 19 20 30 40 42 22 38 44]
```

11. Search and Sort

```
import numpy as np
Scores = np.array([79,45,80,99,90,88,84])
search_score = [84,90]
for Score in search_score :
   indices = np.where(Score == Scores)[0]
   if indices>0:
      print('Indices : ',indices)
   else:
      print('Score Not found')
Ascending = np.sort(Scores)
print(Ascending)
Descending = np.sort(Scores)[::-1]
print(Descending)
```

Output :

```
Indices :  [6]

Indices :  [4]

[45 79 80 84 88 90 99]

[99 90 88 84 80 79 45]
```

12. Filter

```
import numpy as np
prices = np.array([20,40,70,45,90,96,48,10,35])
minmum_price = 20
```

```
maximum_price = 60
filter_Price = [(prices >=minmum_price) & (prices<=maximum_price)]
print('Prices Between the range : ',filter_Price)
```

Output :

```
Prices Between the range :  [array([ True,  True, False,  True, False, False,

True, False,  True])]
```

12. Flatten

```
import numpy as np
array = np.array([[
[1,2,3,4],
[5,6,7,8],
[9,10,11,12]
]])
print(array)
print(array.ndim)
print(array.flatten())
```

Output :

```
[[[ 1  2  3  4]

[ 5  6  7  8]

[ 9 10 11 12]]]

3

[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

13. Encapsulation

```
class BankAccount:
   def __init__(self,initial_balance = 0):
      self.__balance = initial_balance
   def Deposit(self,ammount):
      self.__balance += ammount
   print('Deposit Seccessful')
   def Withdraw(self,ammunt):
      if ammunt < self.__balance:
         self.__balance -= ammunt
         print('Withdraw succesful,New Balance :',self.__balance)
```

```
        else:
            print('Withdrawal amount exceeds current balance')
    def CheckBalance(self):
        return self.__balance
account = BankAccount(initial_balance= 500)
account.Deposit(50)
account.Withdraw(200)
account.Withdraw(20)
```

Output :

Deposit Seccessful

Withdrawal amount exceeds current balance

Withdraw succesful,New Balance : 200

14. Encapsulation 2

```
class LibraryBook:
    def __init__(self, isbn, title, author):
        self.__isbn = isbn
        self._title = title
        self._author = author
        self._status = "available"

    def get_ISBN(self):
        return f"***-{self.__isbn[-4:]}"
    def _display_basic_info(self):
        print(f"Title: {self._title}")
        print(f"Author: {self._author}")

    def borrow_book(self, borrower_name):
        if self._status == "available":
            self._status = "borrowed"
            print(f"The book '{self._title}' has been borrowed by {borrower_name}.")
        else:
            print(f"Sorry, the book '{self._title}' is currently unavailable.")
class DigitalLibraryBook(LibraryBook):
    def __init__(self, isbn, title, author, file_format):
        super().__init__(isbn, title, author)
        self.file_format = file_format
    def display_basic_info(self):
        print("Digital Book Information:")
        self._display_basic_info()
```

```
        print(f"File Format: {self.file_format}")
book = LibraryBook("23-45-79-11", "Science", "Mezbah ")
print("Masked ISBN:", book.get_ISBN())
book.borrow_book("Mithun")
digital_book = DigitalLibraryBook("245-45", "Math", "Enamul","PDF")
digital_book.display_basic_info()
```

Output :

Masked ISBN: ***-9-11

The book 'Science' has been borrowed by Mithun.

Digital Book Information:

Title: Math

Author: Enamul

File Format: PDF