In [1]:

```python
# Exercise 01
from abc import ABC, abstractmethod
class NetworkInterface(ABC):
    @abstractmethod
    def connect(self):
        pass
    @abstractmethod
    def transfer(self):
        pass

class RealNetwork(NetworkInterface):
    def connect(self):
        print("Network is connect")
    def transfer(self):
        print("All Data is tranfser")

R1= RealNetwork()
R1.connect()
R1.transfer()
```

executed in 19ms, finished 23:08:34 2020-07-18

```
Network is connect
All Data is tranfser
```

In [2]:

```python
# Exercise 02
from abc import ABC, abstractmethod
class NetworkInterface(ABC):
    @abstractmethod
    def connect(self):
        pass
    @abstractmethod
    def transfer(self):
        pass

class RealNetwork(NetworkInterface):
    def connect(self):
        print("Network is connect")
    def transfer(self):
        print("All Data is tranfser")

class FakeNetwork(NetworkInterface):
    def connect(self):
        print("Network is connect")

    def transfer(self):
        print("All Data is tranfser")

R1= RealNetwork()
R1.connect()
R1.transfer()
F1= FakeNetwork()
F1.connect()
```

executed in 144ms, finished 23:08:34 2020-07-18

```
Network is connect
All Data is tranfser
Network is connect
```

In [3]:

```python
# Exercise 03
class A:
    def __init__(self, x,y):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return(x,y)

c1 = A(5,3)
c2 = A(2,3)
print("Sum:", c1+c2)
```

executed in 183ms, finished 23:08:34 2020-07-18

```
Sum: (7, 6)
```

In [4]:

```python
# Exercise 04
class A:
    def __init__(self, x):
        self.x = x

    def __lt__(self, other):
        if self.x < other.y:
            return True
        else:
            return False

class B:
    def __init__(self, y):
        self.y = y

c1 = A(2)
c2 = B(4)
print("Is c1 less than c2: ",c1<c2)
```

executed in 146ms, finished 23:08:34 2020-07-18

```
Is c1 less than c2:  True
```

In [5]:

```python
# Exercise 05
class A:
    def __init__(self, a):
        self.a = a

    def __lt__(self, other):
        if self.a < other.a:
            return True
        else:
            return False

    def __eq__(self, other):
        if(self.a == other.a):
            return "Both are equal"
        else:
            return "Not equal"

ob1 = A(2)
ob2 = A(3)
print(ob1 < ob2)

ob3 = A(4)
ob4 = A(4)
print(ob3 == ob4)
```

executed in 111ms, finished 23:08:34 2020-07-18

```
True
Both are equal
```

In [6]:

```python
# Task 01
class A:
    def __init__(self, x):
        self.x = x

    def __gt__(self, other):
        if self.x > other.y:
            return True

        else:
            return False

    def __lt__(self, other):
        if self.x < other.y:
            return True

        else:
            return False

    def __ge__(self, other):
        if self.x >= other.y:
            return True

        else:
            return False

    def __le__(self, other):
        if self.x <= other.y:
            return True

        else:
            return False

    def __eq__(self, other):
        if(self.x == other.y):
            return True
        else:
            return False

    def __ne__(self, other):
        if(self.x != other.y):
            return "Both are equal"
        else:
            return "Not equal"

class B:
    def __init__(self, y):
        self.y = y
```

executed in 113ms, finished 23:08:35 2020-07-18

In [7]:

```
num1 = A(2)
num2 = B(4)
print("Is num1 less than num2? ",num1 > num2)

num1 = A(2)
num2 = B(2)
print("Is num1 is equal to num2? ", num1 == num2)

num1 = A(8)
num2 = B(4)
print("Is num1 is less than or equal to num2? ", num1 <= num2)
```

executed in 117ms, finished 23:08:35 2020-07-18

```
Is num1 less than num2?  False
Is num1 is equal to num2?  True
Is num1 is less than or equal to num2?  False
```

In [8]:

```
# Task 02 Robots Example
from abc import ABC,abstractmethod
# Abstract Class
class Robot(ABC):
    @abstractmethod
    def obeyOrder(self): pass

    @abstractmethod
    def doCleaning(self): pass

# Derived Classes
class Cook(Robot):
    def obeyOrder(self):
        print('Cook Robot is cooking.')
    def doCleaning(self):
        print('Cook Robot is cleaning kitchen.')
    def cooking(self, dish):
        print(f'Robot is cooking {dish}.')
    def baking(self):
        print('Robot is baking cookies.')

class Driver(Robot):
    def obeyOrder(self):
        print('Robot is driving.')
    def doCleaning(self):
        print('Robot is cleaning the vehicle.')
    def drive(self, speed_status):
        print(f'Robot is driving {speed_status}.')
    def driveTo(self, destination):
        print(f'Robot is driving to {destination}')
```

executed in 173ms, finished 23:08:35 2020-07-18

In [9]:

```
1 cook = Cook()
2 cook.obeyOrder()
3 cook.cooking('fish')
4
5 driver = Driver()
6 driver.doCleaning()
7 driver.driveTo('Clifton')
```

executed in 104ms, finished 23:08:35 2020-07-18

```
Cook Robot is cooking.
Robot is cooking fish.
Robot is cleaning the vehicle.
Robot is driving to Clifton
```