In [6]:
```
# Procedural programmming
# functional programming
# Object Oriented Programming Structure(OOPs)


# Procedural programming -> FORTRAN, BASIC, C

username = input('Enter the username: ')
password = input('Enter the password: ')

original_username = 'Piyush' # API (Application Programming Interface)
original_password = '1234' # DB call

if(username == original_username and password == original_password):
    print('Logged in successfully')
else:
    print('Invalid username or password')


username = input('Enter the username: ')
password = input('Enter the password: ')

original_username = 'Piyush' # API (Application Programming Interface)
original_password = '1234' # DB call

if(username == original_username and password == original_password):
    print('Logged in successfully')
else:
    print('Invalid username or password')
```

```
Enter the username: vasi
Enter the password: 1234
Invalid username or password
Enter the username: Piyush
Enter the password: 1234
Logged in successfully
```

In [10]:
```
# functional programming

def validate(username, password):
    original_username = 'Piyush' # API (Application Programming Interface)
    original_password = '1234' # DataBase call

    if(username == original_username and password == original_password):
        print('logged in successfully')
    else:
        print('invalid username or password')

def get_user_input():
    username = input('Enter the username: ')
    password = input('Enter the password: ')
    validate(username, password)

for i in range(3):
    get_user_input()
```

```
Enter the username: dgajhdbas
Enter the password: dadas
invalid username or password
Enter the username: dASDAS
Enter the password: fdafas
invalid username or password
Enter the username: Piyush
Enter the password: 1234
logged in successfully
```

In [13]:
```python
# Object Oriented Programming structure
item = 'Iphone 14'
item_price = 70000
item_quantity = 2
item_total_price = item_price * item_quantity

item2 = 'Charger'
item2_price = 5000
item2_quantity = 1
item2_total_price = item2_price * item2_quantity

print(item, item2)
```

```
Iphone 14 Charger
```

In [16]:
```python
# Object Oriented Programming structure
# variables inside a class -> properties or attributes
# functions inside a class -> methods or behaviours
# class - is a blue print
# object - is an instance(real time entity) of a class

class Human:
    age = 20 # properties
    gender = 'female'
    color = 'white'

    def walk(): # methods
        print('walking')

    def eat():
        print('eating')

vasi = Human()
Ankur = Human()
meena = Human()
print(meena, Human)
```

```
<__main__.Human object at 0x00000298C47AAE50> <class '__main__.Human'>
```

In [30]:
```python
class Item:
    def calculate_total_price(self, price, quantity):
        # print('self --->', self)
        return price * quantity

phone = Item() # instantiation (creating the object -> memories will get allocated)
# print('phone --->', phone)
phone.name = 'Iphone 14'
phone.price = 70000
phone.quantity = 2
```

```
# print(phone.name, phone.price, phone.quantity)
print(phone.calculate_total_price(phone.price, phone.quantity))
```

```
140000
```

In [39]:
```
class Item:
    def calculate_total_price(self, price, quantity):
        # print('self --->', self)
        return price * quantity

phone = Item() # instantiation (creating the object -> memories will get allocated)
# print('phone --->', phone)
phone.name = 'Iphone 14'
phone.price = 70000
phone.quantity = 2

# print(phone.name, phone.price, phone.quantity)
phone_total_price = phone.calculate_total_price(phone.price, phone.quantity)
# print('Phone total price:- ', phone_total_price)

charger = Item()
charger.name = 'Iphone charger'
charger.price = 5000
charger.quantity = 1
charger_total_price = charger.calculate_total_price(charger.price, charger.quantity)
# print('charger total price:- ', charger_total_price)


print(phone.__dict__)
print(charger.__dict__) # magic or dunder methods
```

```
{'name': 'Iphone 14', 'price': 70000, 'quantity': 2}
{'name': 'Iphone charger', 'price': 5000, 'quantity': 1}
```

In [52]:
```
# constructor -> a place where memories are created(initialised) and it is called auto
class Item:
    def __init__(self, name, price, quantity):
#         print('constructor called', self)
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self, price, quantity):
        return price * quantity

phone = Item('Iphone 14', 70000, 2)
charger = Item('Iphone charger', 5000, 1)
# print(phone.name, phone.price, phone.quantity)
# print(phone.__dict__)
# print(charger.__dict__)

phone_total_price = phone.calculate_total_price(phone.price, phone.quantity)
# print(phone_total_price)
```

```
Iphone 14 70000 2
```

In [54]:
```
# constructor -> a place where memories are created(initialised) and it is called auto
class Item:
    def __init__(self, name, price, quantity):
#         print('constructor called', self)
        self.name = name
```

```
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
#         print(self.price, self.quantity)
        return self.price * self.quantity

phone = Item('Iphone 14', 70000, 2)
charger = Item('Iphone charger', 5000, 3)
# print(phone.name, phone.price, phone.quantity)
# print(phone.__dict__)
# print(charger.__dict__)

print(phone.calculate_total_price())
print(charger.calculate_total_price())
```

```
140000
15000
```

In [64]:

```
class Item:
    def __init__(self, name, price, quantity):

        assert price >= 0, f'Invalid price -> {price}. It should be greater than or eq
        assert quantity > 0, f'Invalid quantity -> {quantity}. It should be greater th

        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
        return self.price * self.quantity

phone = Item('Iphone 14', 70000, -2)
print(phone.calculate_total_price())
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [64], in <cell line: 14>()
     11     def calculate_total_price(self):
     12         return self.price * self.quantity
---> 14 phone = Item('Iphone 14', 70000, -2)
     15 print(phone.calculate_total_price())

Input In [64], in Item.__init__(self, name, price, quantity)
      2 def __init__(self, name, price, quantity):
      4     assert price >= 0, f'Invalid price -> {price}. It should be greater than
or equal to zero'
----> 5     assert quantity > 0, f'Invalid quantity -> {quantity}. It should be great
er than zero'
      7     self.name = name
      8     self.price = price

AssertionError: Invalid quantity -> -2. It should be greater than zero
```

In [69]:

```
class Item:
    def __init__(self, name, price = 0, quantity = 1):

        assert price >= 0, f'Invalid price -> {price}. It should be greater than or eq
        assert quantity > 0, f'Invalid quantity -> {quantity}. It should be greater th
```

```
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
        return self.price * self.quantity

phone = Item('Iphone 14', 70000, 3)
print(phone.calculate_total_price())
```

```
210000
```

In [75]:
```
class Item:
    def __init__(self, name, price = 0, quantity = 1):

        assert price >= 0, f'Invalid price -> {price}. It should be greater than or ec
        assert quantity > 0, f'Invalid quantity -> {quantity}. It should be greater th

        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
        return self.price * self.quantity

phone = Item('Iphone 14', 70000, 3)
phone.calculate_total_price()



name = 'vasanth'
print(name.upper(), type(name))

num = 10
print(type(num), num)

names = ['virat', 'rohit']
print(type(names), names.pop())
```

```
VASANTH <class 'str'>
<class 'int'> 10
<class 'list'> rohit
```

In [76]:
```
name = 'vasanth'
print(name)

name = str('rajeev')
print(name)

names = tuple()
```

```
vasanth
rajeev
```

In [82]:
```
class Item:
    def __init__(self, name, price = 0, quantity = 1):

        assert price >= 0, f'Invalid price -> {price}. It should be greater than or ec
        assert quantity > 0, f'Invalid quantity -> {quantity}. It should be greater th
        # object / self properties
        self.name = name
```

```
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
        return self.price * self.quantity

phone = Item('Iphone 14', 70000, 3)
charger = Item('Iphone charger', 5000, 1)
# print(phone.price)

phone.price = 60000
# phone.quantity = 2
print(phone.price, charger.price)
```

```
60000 5000
```

In [89]:

```
class Item:
    # class property - constant for all objects
    discount = 0.2 # 20% discount

    def __init__(self, name, price = 0, quantity = 1):

        assert price >= 0, f'Invalid price -> {price}. It should be greater than or ec
        assert quantity > 0, f'Invalid quantity -> {quantity}. It should be greater th
        # object / self properties
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price - (self.price * Item.discount)

phone = Item('Iphone 14', 70000, 1)
charger = Item('Iphone charger', 5000, 1)

phone.apply_discount()
print(phone.calculate_total_price())

charger.apply_discount()
print(charger.calculate_total_price())
```

```
56000.0
4000.0
```

In [95]:

```
class Item:
    # class property - constant for all objects
    discount = 0.2 # 20% discount

    def __init__(self, name, price = 0, quantity = 1):

        assert price >= 0, f'Invalid price -> {price}. It should be greater than or ec
        assert quantity > 0, f'Invalid quantity -> {quantity}. It should be greater th
        # object / self properties
        self.name = name
        self.price = price
        self.quantity = quantity
```

```
    def calculate_total_price(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price - (self.price * self.discount) # by default it will ta

phone = Item('Iphone 14', 70000, 1)
charger = Item('Iphone charger', 5000, 1)

phone.discount = 0.5
phone.apply_discount()
print(phone.calculate_total_price())

charger.discount = 0.8
charger.apply_discount()
print(charger.calculate_total_price())
```

```
35000.0
1000.0
```

In [101…

```
class Item:
    # class property - constant for all objects
    discount = 0.2 # 20% discount

    def __init__(self, name, price = 0, quantity = 1):

        assert price >= 0, f'Invalid price -> {price}. It should be greater than or eq
        assert quantity > 0, f'Invalid quantity -> {quantity}. It should be greater th
        # object / self properties
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price - (self.price * self.discount) # by default it will ta

phone = Item('Iphone 14', 70000, 1)
# additional properties can be created
phone.is_case_available = False
print(phone.__dict__)

samsung = Item('samsung 22', 50000, 1)
samsung.is_case_available = True
print(samsung.__dict__)

tomato = Item('tomato', 30, 1)
print(tomato.__dict__)
```

```
{'name': 'Iphone 14', 'price': 70000, 'quantity': 1, 'is_case_available': False}
{'name': 'samsung 22', 'price': 50000, 'quantity': 1, 'is_case_available': True}
{'name': 'tomato', 'price': 30, 'quantity': 1}
```

In [ ]:

```
# OOPs -> 4 pillars -> Encapsulation, Abstraction, Inheritance and Polymorphism
```

In [113…

```
# ATM
class ATM:
    original_username = 'vaishnavi'
```

```
    original_pin_number = '1234'

    def __init__(self, username, pin_number):
        self.username = username
        self.pin_number = pin_number

    def validate(self):
        if(self.username == ATM.original_username and self.pin_number == ATM.original_
            return True
        else:
            return False

Meena = ATM('meena', '6789')
# print(Meena.__dict__)
# print(Meena.original_username, Meena.original_pin_number)
# Meena.original_username = 'meena'
# Meena.original_pin_number = '6789'
# ATM.original_username = 'meena'
# ATM.original_pin_number = '6789'
print(Meena.validate())
```

```
True
```

In [125…

```
# Encapsulation --> public, private(can't be accessible outside class)
class ATM:
    __original_username = 'vaishnavi' # to change a property to private, add 2 _
    __original_pin_number = '1234'

    def __init__(self, username, pin_number):
        self.username = username
        self.pin_number = pin_number

    def validate(self):
        if(self.username == ATM.__original_username and self.pin_number == ATM.__origi
            return True
        else:
            return False

Meena = ATM('meena', '6789')
# print(ATM.__original_username, ATM.__original_pin_number)
Vaishnavi = ATM('vaishnavi', '1234')
print(Vaishnavi.__original_username, Vaishnavi.__original_pin_number)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [125], in <cell line: 19>()
     17 # print(ATM.__original_username, ATM.__original_pin_number)
     18 Vaishnavi = ATM('vaishnavi', '1234')
---> 19 print(Vaishnavi.__original_username, Vaishnavi.__original_pin_number)

AttributeError: 'ATM' object has no attribute '__original_username'
```

In [ ]:

```
# getters and setters
```