**Write a class called Product. The class should have fields called name, amount, and price, holding the product's name, the number of items of that product in stock, and the regular price of the product. There should be a method get_price that receives the number of items to be bought and returns a the cost of buying that many items, where the regular price is charged for orders of less than 10 items, a 10% discount is applied for orders of between 10 and 99 items, and a 20% discount is applied for orders of 100 or more items. There should also be a method called make_purchase that receives the number of items to be bought and decreases amount by that much. ¶**

In [1]:
```
class Product:

    def __init__(self, name, amount, price):
        self.name = name
        self.amount = amount
        self.price = price

    def get_price(self, number_to_be_bought):
        discount = 0
        if number_to_be_bought < 10:
            pass
        elif 10 <= number_to_be_bought < 99:
            discount = 10
        else:
            discount = 20
        price = (100 - discount) / 100 * self.price
        return price * number_to_be_bought

    def make_purchase(self, quantity):
        self.amount -= quantity


# name = input('name:')
# amount = int(input('Digit amount of items'))
# price = int(input('Digit price of items'))

name, amount, price = 'shoes', 200, 33

shoes = Product(name, amount, price)
# quantity = int(input('Digit amount of items to buy'))

q1 = 4
print(f'cost for {q1} {shoes.name} = {shoes.get_price(q1)}')
shoes.make_purchase(q1)
print(f'remaining stock: {shoes.amount}\n')

q2 = 12
print(f'cost for {q2} {shoes.name} = {shoes.get_price(q2)}')
shoes.make_purchase(q2)
print(f'remaining stock: {shoes.amount}\n')

q3 = 112
print(f'cost for {q3} {shoes.name} = {shoes.get_price(q3)}')
shoes.make_purchase(q3)
print(f'remaining stock: {shoes.amount}\n')
```

```
cost for 4 shoes = 132.0
remaining stock: 196

cost for 12 shoes = 356.4
remaining stock: 184

cost for 112 shoes = 2956.8
remaining stock: 72
```

# You need to create the foundations of an e-commerce engine for a B2C (business-to-consumer) retailer. You need to have a class for a customer called User, a class for items in inventory called Item, and a shopping cart class called Cart. Items go in Carts, and Users can have multiple Carts. Also, multiple items can go into Carts, including more than one of any single item.

In [6]:
```
class User:
    def __init__(self, id, name):
        self.id = id
        self.name = name
    def display_user(self):
        print('ID', self.id, 'Name:', self.name)

class Item:
    def __init__(self, id, name, price, sold, available):
        self.id = id
        self.name = name
        self.price = price
        self.sold = sold
        self.available = available

class Cart:
    def __init__(self, user):
        self.user = user
        self.cart_items = []
    def insert_items(self, item, quantity):
        for i in range(quantity):
            if item.available == 0:
                print('Out of stock')
                break
            self.cart_items.append(item)
            item.sold += 1
            item.available -= 1
    def display_cart(self):
        print('This Cart belongs to', self.user.name, 'with ID', self.user.id)
        self.total = 0
        for i in self.cart_items:
            print('Item',i.name)
            self.total += i.price
        print('Total price = ', self.total)



#Creating a user
user1 = User(1, 'Kavit')
user1.display_user()

#Creating items
apple = Item(1, 'apple', 100, 0, 10)
greencoconut = Item(2, 'green coconut', 150, 0, 1)
milk = Item(2, 'milk', 24, 0, 100)

#Creating two carts for user1
cart1 = Cart(user1)
cart2 = Cart(user1)

#Adding items to cart1
cart1.insert_items(apple, 2)
cart1.insert_items(milk, 3)

#Adding items to cart2
cart2.insert_items(greencoconut, 3)
```

```
cart2.insert_items(milk, 20)

#Displaying details of cart1 and cart2
cart1.display_cart()
cart2.display_cart()
```

```
ID 1 Name: Kavit
Out of stock
This Cart belongs to Kavit with ID 1
Item apple
Item apple
Item milk
Item milk
Item milk
Total price =  272
This Cart belongs to Kavit with ID 1
Item green coconut
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Total price =  630
```

In [11]:
```
class Pizza:
    def __init__(self, size, toppings, cheese):
        self.size = size
        self.toppings = toppings
        self.cheese = cheese

    def price(self):
        self.cost = 0
        if self.size == 'small':
            self.cost += 50
        elif self.size == 'medium':
            self.cost += 100
        else:
            self.cost += 200
        topping_prices_20 = ['corn', 'tomato', 'onion', 'capsicum']
        topping_prices_50 = ['mushroom', 'olives', 'broccoli']
        for topping in self.toppings:
            if topping in topping_prices_20:
                self.cost += 20
            else:
                self.cost += 50
        #for cheese
        self.cost += 50 * len(self.cheese)
        return self.cost

class Order:
    def __init__(self, name, customerid):
        self.name = name
        self.customerid = customerid

    def order(self, n):
        self.pizzas = []
        for i in range(n):
            toppings = []
            cheese = []
            print('Customize Pizza',i+1)
            size = input('Select size: ')
            t = int(input('How many toppings: '))
            for i in range(t):
                toppings.append(input('Enter toppings: '))
            t = int(input('How many cheese: '))
            for i in range(t):
                cheese.append(input('Enter cheese: '))
            self.pizzas.append(Pizza(size, toppings, cheese))

    def bill(self):
        self.total = 0
        count = 1
        for p in self.pizzas:
            print('Pizza', count)
            print(p.size, p.toppings, p.cheese)
            self.total += p.price()
            count += 1
        print('Total bill amount:', self.total)

number=int(input("How many pizzas you want to order: "))
```

```
order1 = Order('Kavit', 1)
order1.order(number)
order1.bill()
```

```
How many pizzas you want to order: 2
Customize Pizza 1
Select size: small
How many toppings: 2
Enter toppings: olives
Enter toppings: capsicum
How many cheese: 2
Enter cheese: feta
Enter cheese: cheddar
Customize Pizza 2
Select size: medium
How many toppings: 1
Enter toppings: corn
How many cheese: 1
Enter cheese: processed
Pizza 1
small ['olives', 'capsicum'] ['feta', 'cheddar']
Pizza 2
medium ['corn'] ['processed']
Total bill amount: 390
```

## Write a program to build a simple Student Management System using Python which can perform the following operations:

1. Accept
2. Display
3. Search
4. Delete
5. Update

### Accept – This method takes details from the user like name, roll number, and marks for two different subjects.

### Display – This method displays the details of every student.

### Search – This method searches for a particular student from the list of students. This method will ask the user for roll number and then search according to the roll number

### Delete – This method deletes the record of a particular student with a matching roll number.

### Update – This method updates the roll number of the student. This method will ask for the old roll number and new roll number. It will replace the old roll number with a new roll number.

In [20]:
```
# This is simplest Student data management program in python

# Create class "Student"
class Student:

# Constructor
    def __init__(self, name, rollno, m1, m2):
        self.name = name
        self.rollno = rollno
        self.m1 = m1
        self.m2 = m2

    # Function to create and append new student
    def accept(self):
        Name=input("Enter Name: ")
        Rollno=int(input("Enter Rollno.: "))
        marks1=int(input("Enter marks of 1: "))
        marks2=int(input("Enter marks of 2: "))
        ob = Student(Name, Rollno, marks1, marks2)
        ls.append(ob)

    # Function to display student details
    def display(self, ob):
        print("Name : ", ob.name)
        print("RollNo : ", ob.rollno)
        print("Marks1 : ", ob.m1)
        print("Marks2 : ", ob.m2)
        print("\n")

    # Search Function
    def search(self, rn):
        for i in range(ls.__len__()):
            if(ls[i].rollno == rn):
                return i

    # Delete Function
    def delete(self, rn):
        i = obj.search(rn)
        del ls[i]

    # Update Function
    def update(self, rn, No):
        i = obj.search(rn)
        roll = No
        ls[i].rollno = roll


# Create a list to add Students
ls = []
# an object of Student class
obj = Student('', 0, 0, 0)

print("\nOperations used, ")
print("\n1.Accept Student details\n2.Display Student Details\n3.
      Search Details of a Student\n4.Delete Details of Student\n5.Update Student
```

```
# ch = int(input("Enter choice:"))
# if(ch == 1):
obj.accept()
obj.accept()
obj.accept()

# elif(ch == 2):
print("\n")
print("\nList of Students\n")
for i in range(ls.__len__()):
    obj.display(ls[i])

# elif(ch == 3):
print("\n Student Found, ")
s = obj.search(10)
obj.display(ls[s])

# elif(ch == 4):
obj.delete(30)
print(ls.__len__())
print("List after deletion")
for i in range(ls.__len__()):
    obj.display(ls[i])

# elif(ch == 5):
obj.update(20,15)
print(ls.__len__())
print("List after updation")
for i in range(ls.__len__()):
    obj.display(ls[i])

# else:
print("Thank You !")
```

```
Operations used,

1.Accept Student details
2.Display Student Details
3.Search Details of a Student
4.Delete Details of Student
5.Update Student Details
6.Exit
Enter Name: Kavit
Enter Rollno.: 10
Enter marks of 1: 80
Enter marks of 2: 90
Enter Name: Vishal
Enter Rollno.: 20
Enter marks of 1: 78
Enter marks of 2: 88
Enter Name: Manish
Enter Rollno.: 30
Enter marks of 1: 75
Enter marks of 2: 89
```

```
List of Students

Name :  Kavit
RollNo :  10
Marks1 :  80
Marks2 :  90


Name :  Vishal
RollNo :  20
Marks1 :  78
Marks2 :  88


Name :  Manish
RollNo :  30
Marks1 :  75
Marks2 :  89



 Student Found,
Name :  Kavit
RollNo :  10
Marks1 :  80
Marks2 :  90


2
List after deletion
Name :  Kavit
RollNo :  10
Marks1 :  80
Marks2 :  90


Name :  Vishal
RollNo :  20
Marks1 :  78
Marks2 :  88


2
List after updation
Name :  Kavit
RollNo :  10
Marks1 :  80
Marks2 :  90


Name :  Vishal
RollNo :  15
Marks1 :  78
Marks2 :  88
```

```
Thank You !
```

## Stacks and Queues. Write a class which defines a data structure that can behave as both a queue (FIFO) or a stack (LIFO), There are four methods that should be implemented:

### shift() returns the first element and removes it from the list

### unshift() "pushes" a new element to the front or head of the list

### push() adds a new element to the end of a list

### pop() returns the last element and removes it from the list

In [24]:
```
class StackQueue:
    def __init__(self, L):
        self.L = L
    def shift(self):
        if len(self.L) == 0:
            pass
            raise Exception('List is empty')
        try:
            x = self.L.pop(0)
            return x
        except Exception:
            print(Exception)
    def unshift(self, n):
        self.L.insert(0, n)
    def push(self, n):
        self.L.append(n)
    def pop(self):
        if len(self.L) == 0:
            raise Exception('List is empty')
        try:
            x = self.L.pop()
            return x
        except Exception:
            print(Exception)
    def display(self):
        return self.L

sq = StackQueue([1,4,6,8,9])
print(sq.shift())
sq.unshift(7)
sq.push(10)
print(sq.pop())
print(sq.display())
```

```
1
10
[7, 4, 6, 8, 9]
```