**Задание 1**. Создайте класс Batary, у которой будет определен атрибут capacity = [ ] (емкость), max\_charge = 5 (максимальный заряд) по умолчанию, и методы: charge - заряжает батарею и discharge - разряжает батарею.

Переопределите метод \_\_str\_\_, чтобы при вызове экземпляра он представлялся в виде: [)))))] - максимально заряженная батарея.

class Battery:  
 def \_\_init\_\_(self):  
 self.capacity=[]  
 self.max\_charge=5  
 for i in range(self.max\_charge):  
 self.capacity.append(')')  
  
 def charge(self, n):  
 if len(self.capacity)+n>self.max\_charge:  
 self.capacity=[')' for i in range(self.max\_charge)]  
 else:  
 for i in range(n):  
 self.capacity.append(')')  
  
 def discharge(self, n):  
 if len(self.capacity)-n<0:  
 self.capacity=['(' for i in range(self.max\_charge)]  
 else:  
 for i in range(n):  
 self.capacity.pop()  
  
 def \_\_str\_\_(self):  
 return ''.join(self.capacity)  
  
  
b=Battery()  
print(b)  
b.discharge(4)  
print(b)  
b.charge(2)  
print(b)

**Результат:**

)))))

)

)))

**Задание 2.** Реализовать класс Queue. Определить атрибут inside, который будет хранить в себе имена людей в очереди. Переопределить метод \_\_str\_\_, чтобы преобразовать его к виду: Name1=>Name2=>...=>Name3

Реализовать методы: add - который добавляет имя в очередь, take\_out - убирает первого человека из списка. Переопределить методы \_\_add\_\_ , \_\_sub\_\_, \_\_iadd\_\_, \_\_isub\_\_ чтобы они соответствовали методам add и take\_out.

class Queue:  
 def \_\_init\_\_(self):  
 self.inside=["Оля", "Лера"]  
 def add(self, name):  
 self.inside.append(name)  
 def take\_out(self):  
 if len(self.inside)>0:  
 self.inside.pop(0)  
 def \_\_str\_\_(self):  
 return '=>'.join(self.inside)  
 def \_\_add\_\_(self, name):  
 self.add(name)  
 return self  
 def \_\_sub\_\_(self, name):  
 self.take\_out()  
 return self  
 def \_\_iadd\_\_(self, name):  
 self.add(name)  
 return self  
 def \_\_isub\_\_(self, name):  
 self.take\_out()  
 return self  
  
q=Queue()  
q.add("Коля")  
print(q)  
q.take\_out()  
print(q)

**Результат:** Оля=>Лера=>Коля

Лера=>Коля

**Задание 3.** a) Создайте класс Matrix**,** который должен принимать данные (список списков) для формирования матрицы.

Пример ввода:

m1 = Matrix([[1,2,3], [4,5,6]])

примеры матриц:

0 0 7 8 9

0 0 10 3 -1

12 3 4

b) Следующий шаг - реализовать перегрузку метода \_\_str\_\_() для вывода матрицы в привычном виде.

c) Далеее реализовать перегрузку метода \_\_add\_\_() для сложения двух объектов класса Matrix (двух матриц). Результатом сложения должна быть новая матрица.

Пример: m1 = Matrix([[1,2,3],[4,5,6]])

m2 = Matrix([[6,5,4],[3,2,1]])

m3 = m1 + m2

print(m3)

class Matrix:  
 def \_\_init\_\_(self, data):  
 self.data=data  
 def \_\_str\_\_(self):  
 output=""  
 for row in self.data:  
 output+=" ".join([str(elem) for elem in row])  
 output+="\n"  
 return output  
 def \_\_add\_\_(self, other):  
 new\_data=[]  
 for row1, row2 in zip(self.data, other.data):  
 new\_row=[]  
 for elem1, elem2 in zip(row1, row2):  
 new\_row.append(elem1 + elem2)  
 new\_data.append(new\_row)  
 return Matrix(new\_data)  
  
m1=Matrix([[8, 3, 5], [1, 5, 2]])  
print(m1)  
m2=Matrix([[1, 3, 1], [4, 2, 3]])  
print(m2)  
m3=m1+m2  
print(m3)

**Результат:**

8 3 5

1 5 2

1 3 1

4 2 3

9 6 6

5 7 5

**Задание 4 \* (необязательное).** Реализуйте класс Deque - двух стороннюю очередь или колоду. Идея та же, что и у очереди, только можно вставлять элементы в центр массива убирать как слева так и справа.

1) Реализуйте класс Deque

2) Переопределите метод \_\_str\_\_

3) Реализуйте методы, который позволят вставлять элементы слева, справа, в центр массива и аналогично удалять.

class Deque:  
 def \_\_init\_\_(self):  
 self.items=[]  
 def \_\_str\_\_(self):  
 return str(self.items)  
 def add\_front(self, item):  
 self.items.insert(0, item)  
 def add\_rear(self, item):  
 self.items.append(item)  
 def remove\_front(self):  
 if self.items:  
 return self.items.pop(0)  
 else:  
 return None  
 def remove\_rear(self):  
 if self.items:  
 return self.items.pop()  
 else:  
 return None  
  
d=Deque()  
d.add\_front(1)  
d.add\_rear(2)  
d.add\_rear(4)  
print(d)  
d.remove\_front()  
d.remove\_rear()  
print(d)

**Результат:**

[1, 2, 4]

[2]