Задачи

Вспомните про unittest и по возможности оформите тесты как юниттесты.

Реализуйте класс Matrix.

Вся матрица должна хранится в виде 1 (ОДНОГО ОДНОМЕРНОГО) списка.

Он должен содержать:

Конструктор

Конструктор от списка списков. Гарантируется, что списки состоят из чисел, не пусты и все имеют одинаковый размер. Конструктор должен копировать содержимое списка списков, т.е. при изменении списков, от которых была сконструирована матрица, содержимое матрицы изменяться не должно.

__str__

Метод __str__ переводящий матрицу в строку. При этом элементы внутри одной строки должны быть разделены знаками табуляции, а строки — переносами строк. При этом после каждой строки не должно быть символа табуляции и в конце не должно быть переноса строки.

size

Метод size без аргументов, возвращающий кортеж вида (число строк, число столбцов)

reshape

Устанавливает новые параметры количества строк и столбцов. С проверкой на корректность.

```
Тест 1
Входные данные:
# Task 1 check 1
m = Matrix([[1, 0], [0, 1]])
print(m)
m = Matrix([[2, 0, 0], [0, 1, 10000]])
print(m)
m = Matrix([[-10, 20, 50, 2443], [-5235, 12, 4324, 4234]])
print(m)

Вывод программы:
```

```
1    0
0    1
2    0    0
0    1    10000
-10    20    50    2443
-5235    12    4324    4234
```

Тест 2
Входные данные:

```
# Task 1 check 2
m1 = Matrix([[1, 0, 0], [1, 1, 1], [0, 0, 0]])
m2 = Matrix([[1, 0, 0], [1, 1, 1], [0, 0, 0]])
print(str(m1) == str(m2))
```

Вывод программы:

```
True
```

Тест 3
Входные данные:

```
# Task 1 check 3
m = Matrix([[1, 1, 1], [0, 100, 10]])
print(str(m) == '1\t1\t1\n0\t100\t10')
```

Вывод программы:

```
True
```

сложение и умножение

- __add__ принимающий вторую матрицу того же размера и возвращающий сумму матриц
- __mul__ принимающий число типа int или float и возвращающий матрицу, умноженную на скаляр
- __rmul__ делающий то же самое, что и __mul__. Этот метод будет вызван в том случае, аргументнаходится справа. Можно написать __rmul__ = __mul__

В этом случае вызовется __mul__: Matrix([[0, 1], [1, 0]) *10 В этом случае вызовется __rmul__ (так как у int не определен __mul__ для матрицы справа):* *10* Matrix([[0, 1], [1, 0])

Разумеется, данные методы не должны менять содержимое матрицы.

Тест 1
Входные данные:

```
# Task 2 check 1
m = Matrix([[10, 10], [0, 0], [1, 1]])
print(m.size())
```

Вывод программы:

```
(3, 2)
```

Тест 2
Входные данные:

```
# Task 2 check 2
m1 = Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
m2 = Matrix([[0, 1, 0], [20, 0, -1], [-1, -2, 0]])
print(m1 + m2)
Вывод программы:
1    1    0
20    1    -1
-1    -2    1



Тест 3
Входные данные:
# Task 2 check 3
m = Matrix([[1, 1, 0], [0, 2, 10], [10, 15, 30]])
alpha = 15
print(m * alpha)
print(alpha * m)
Вывод программы:
15    15    0
0    30    150
150    225    450
15    15    0
0    30    150
150    225    450
```

__add__ - проверка корректности

Добавьте в программу из предыдущей задачи класс MatrixError, содержащий внутри self поля matrix1 и matrix2 (ссылки на матрицы).

Добавьте в метод **add** проверку на ошибки в размере входных данных, чтобы при попытке сложить матрицы разных размеров было выброшено исключение MatrixError таким образом, чтобы matrix1 поле MatrixError стало первым аргументом **add** (просто self), а matrix2 — вторым (второй операнд для сложения).

транспонирование

- Реализуйте метод transpose, транспонирующий матрицу и возвращающую результат (данный метод модифицирует экземпляр класса Matrix)
- Реализуйте статический метод transposed, принимающий Matrix и возвращающий транспонированную матрицу.

Подумайте, надо ли делать транспонирование методом класса или статическим методом?

```
Тест 1
Входные данные:
# Task 3 check 1
```

```
# Check exception to add method
m1 = Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
m2 = Matrix([[0, 1, 0], [20, 0, -1], [-1, -2, 0]])
print(m1 + m2)

m2 = Matrix([[0, 1, 0], [20, 0, -1]])
try:
    m = m1 + m2
    print('WA\n' + str(m))
except MatrixError as e:
    print(e.matrix1)
    print(e.matrix2)
Вывод программы:
1    1    0
20    1    -1
-1    -2    1
1    0    0
0    1    0
0    0    1
0    1    0
20    0    -1



Тест 2
Входные данные:
# Task 3 check 2
m = Matrix([[10, 10], [0, 0], [1, 1]])
print(m)
m1 = m.transpose()
print(m)
print(m1)
Вывод программы:
10    10
0    0
1    1
10    0    1
10    0    1
10    0    1
10    0    1



Тест 3
Входные данные:
# Task 3 check 3
m = Matrix([[10, 10], [0, 0], [1, 1]])
print(m)
print(Matrix.transposed(m))
print(m)
Вывод программы:
10    10
0    0
1    1
10    0    1
```

```
10    0    1
10    10
0    0
1    1
```

solve

Пусть экземпляр класса Matrix задаёт систему линейных алгебраических уравнений.

Тогда добавьте в класс метод solve, принимающий вектор-строку свободных членови возвращающий строку-список, состоящую из float — решение системы, если оно единственно. Если решений нет или оно не единственно — выдайте какую-нибудь ошибку.

```
Тест 1
Входные данные:
# Task 5 check 1
m = Matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
print(m.solve([1,1,1]))
Вывод программы:
[1.0, 1.0, 1.0]



Тест 2
Входные данные:
# Task 5 check 2
m = Matrix([[1, 1, 1], [0, 2, 0], [0, 0, 4]])
print(m.solve([1,1,1]))
Вывод программы:
[0.25, 0.5, 0.25]



Тест 3
Входные данные:
# Task 5 check 3
m = Matrix([[1, 1, 1], [0, 1, 2], [0.5, 1, 1.5]])
try:
    s = m.solve([1,1,1])
    print('WA No solution')
except Exception as e:
    print('OK')
Вывод программы:
OK
```

возведение в степень

К программе в предыдущей задаче добавьте класс SquareMatrix — наследник Matrix с операцией возведенияв степень __pow__, принимающей натуральную

(с нулём) степень, в которую нужно возвести матрицу. Используйте быстрое возведение в степень.

Тест 1
Входные данные:
```
# Task 6 check 1
m = SquareMatrix([[1, 0], [0, 1]])
print(isinstance(m, Matrix))
```
Вывод программы:
```
True
```

Тест 2
Входные данные:
```
# Task 6 check 2
m = SquareMatrix([[1, 0], [0, 1]])
print(m ** 0)
```
Вывод программы:
```
1    0
0    1
```

Тест 3
Входные данные:
```
# Task 6 check 3
m = SquareMatrix([[1, 1, 0, 0, 0, 0],
                  [0, 1, 1, 0, 0, 0],
                  [0, 0, 1, 1, 0, 0],
                  [0, 0, 0, 1, 1, 0],
                  [0, 0, 0, 0, 1, 1],
                  [0, 0, 0, 0, 0, 1]]
                 )
print(m)
print('----------')
print(m ** 1)
print('----------')
print(m ** 2)
print('----------')
print(m ** 3)
print('----------')
print(m ** 4)
print('----------')
print(m ** 5)
```
Вывод программы:
```
1    1    0    0    0    0
0    1    1    0    0    0
0    0    1    1    0    0
0    0    0    1    1    0
0    0    0    0    1    1
0    0    0    0    0    1
----------
1    1    0    0    0    0
0    1    1    0    0    0
```

```
0    0    1    1    0    0
0    0    0    1    1    0
0    0    0    0    1    1
0    0    0    0    0    1
----------
1    2    1    0    0    0
0    1    2    1    0    0
0    0    1    2    1    0
0    0    0    1    2    1
0    0    0    0    1    2
0    0    0    0    0    1
----------
1    3    3    1    0    0
0    1    3    3    1    0
0    0    1    3    3    1
0    0    0    1    3    3
0    0    0    0    1    3
0    0    0    0    0    1
----------
1    4    6    4    1    0
0    1    4    6    4    1
0    0    1    4    6    4
0    0    0    1    4    6
0    0    0    0    1    4
0    0    0    0    0    1
----------
1    5    10    10    5    1
0    1    5    10    10    5
0    0    1    5    10    10
0    0    0    1    5    10
0    0    0    0    1    5
0    0    0    0    0    1
```

Uno card game (на дом)

Добавить карты +2, skip, wild, wild+4