Matriz

Neste exercício, definiremos uma classe chamada `Matrix` para representar matrizes bidimensionais. O método `__init__` deve receber uma lista de listas como entrada, onde cada lista interna representa uma única linha na matriz. Por exemplo, `Matrix([[1,2,3], [4,5,6]])` representa a matriz:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Você é livre para usar esta estrutura(listas aninhadas) como sua representação interna ou pode usar qualquer outra representação que você imaginar que faça sentido. Sua representação interna não será representada.

Sua classe deve fornecer os seguintes métodos:

- um método `size()`, que retorna uma tupla `(numrows, numcols)`, onde `numrows` é o número de linhas na matriz e `numcols` é o número de colunas na matriz.
- um método `get(r, c)`, que retorna o número na r-ésima linha e c-ésima coluna, onde as linhas e colunas são indexadas a partir de zero (por exemplo, `m.get(0,0)` deve obter o valor na primeira linha de `m`, primeira coluna).
- um método `set(r, c, val)`, que define o elemento na r-ésima linha e c-ésima coluna para `val`. Este método deve *modificar* a instância, mas não deve retornar nada.
- um método `row(n)`, que retorna uma lista contendo os elementos na n-ésima linha da matriz, onde as linhas são novamente indexadas a partir de zero.
- um método `col(n)`, que retorna uma lista contendo os elementos na n-ésima coluna da matriz, onde as colunas são novamente indexadas a partir de zero.
- um método `transpose()`, que retorna uma *nova instância* de `Matrix` que representa a transposta (https://pt.wikipedia.org/wiki/Matriz_transposta) do original. Este método não deve modificar a instância original de `Matrix`.
- um método `add(other)`, que retorna uma nova instância de `Matrix` representando a soma da instância original e `other`:
  - se `other` for uma instância de `Matrix` com dimensões apropriadas, seu código deve realizar uma adição de matriz (https://pt.wikipedia.org/wiki/Adi%C3%A7%C3%A3o_de_matrizes)
  - se `other` for uma instância de `Matrix` cujas dimensões são inadequadas para adição de matriz, seu código deve retornar `None`
  - se `other` for um `int` ou `float`, adicione `other` a cada entrada
  - caso contrário, retorne `None` (indicando um erro)
- um método `sub(other)`, que deve se comportar de forma análoga a `add`, mas deve realizar a subtração.
- um método `mul(other)`, que retorna uma nova instância de `Matrix` representando o resultado da multiplicação da instância original e `other` (`self` × `other`):
  - se `other` for uma instância de `Matrix` com dimensões apropriadas, seu código deve realizar uma multiplicação de matriz (https://pt.wikipedia.org/wiki/Produto_de_matrizes).
  - se `other` for uma instância de `Matrix` cujas dimensões não permitem uma multiplicação de matriz, seu código deve retornar `None`.
  - se `other` for um `int` ou `float`, multiplique cada entrada por `other`
  - caso contrário, retorna `None` (indicando um erro)

Observe que `add`, `sub` e `mul` *não* devem modificar a instância original de `Matrix`, mas sim criar uma nova instância.

Você também deve implementar o seguinte, que substitui alguns dos comportamentos padrão do Python:

- `__add__` deve ser sobrescrito para que `m1 + o` execute `m1.add(o)`.
- `__sub__` deve ser sobrescrito para que `m1 - o` execute `m1.sub(o)`.
- `__mul__` deve ser sobrescrito para que `m1 * o` execute `m1.mul(o)`.

Submissão

Quando estiver pronto(depois de ter simulado manualmente e testado em sua própria máquina e estiver convencido de que seu programa fará a coisa certa), faça upload do seu arquivo Python no **Problema 5.2** no Gradescope. Lembre de nomear seu arquivo `p5_2.py`.