Home work 2 and 3

Policy on plagiarism

These are individual homework. While you may discuss the ideas and algorithms or share the test cases with others, at no time may you read, possess, or submit the solution code of anyone else (including people outside this course), submit anyone else's solution code to your account on the gradebot, or allow anyone else to read or possess your source code. We will detect plagiarism using automated tools and will prosecute all violations to the fullest extent of the university regulations, including failing this course, academic probation, and expulsion from the university.

Goal

- Matrix calcluation
- Class
- Standard operators, e.g., `+`
- Recursion
- Stack for homework 3
- Binary Tree for homework 3

Description

Matrices are widely used both in mathematics and computer sciences, such as game thoery and economics. In homework 2 and 3, you will implement a module for matrix calculation.

Background

- Matrix Wiki
- Prefix notation (Polish notation)
- Postfix notation (Reverse notation)
- Invertible matrix
- Determinant

Syntax of Matrices

- `Matrix` is defined in Backus-Naur Form, see BNF Wiki.
- In case you don't know matrix or linear algbra well, search on internet，it is not too diffcult to understand. Programs are used to solve real-world problems. However, to

understand the problems, you may have to learn some domain knowledge related to the problems.

```

```
Matrix::= "[" Rows "]" | "[" "]"

Rows::= Row | Row ";" Rows

Row::= element | element "," Row

elment::= int | float | complex
```

```

Howework 2

- Implement a class `Matrix` for representing matrix and implement the following operations in the Matrix class. Instance object of Matrix is created by `x=Matrix(s)`, where `s` is a string in **Matrix** syntax allowing white space `" "`. If the string `s` does not follow the **Matrix** syntax, raise the exception `MatrixSyntaxError`
- Operations: `+`, `-`, `*`, `/`, `**`, `==`, `isIdentity`, `isSquare`, `determinant`, `inverse`, `index`, `slice`, `__str__`, `transposition`
  - `+, -, *, /, **, ==` are standard operators, like for `int, float`. `*` supports both Scalar multiplication `x * c` and Matrix Multiplication `x * y`. For division and pow, we only condsider `x/c` and `x**c`. Note that `x,y` are matrices and `c` is a number (`int, float, complex`). The operands of `+,-,==`are matrices. Raise the exception `MatrixSyntaxError` if the operands of `+`, `-`, `*`, `**` do not meet the requirment for the corresponding operation, e.g., adding a `2-by-3` matrix with a `2-by-4` matrix should raise the exception
  - `isIdentity` returns `True` if the matrix is an identity matrix, otherwise `False`, empty matrix is regarded as an identity matrix
  - `isSquare` returns `True` if the matrix is a square matrix, otherwise `False`, empty matrix is regarded as a square matrix
  - `index` similar to list index
  1. `x[i]` returns the `i`-th row (starting at `0`) which also is a matrix
  2. `x[i,j]` returns the element at the `i`-th row and `j`-th column
  3. `x[i,j]=1` replaces the element `x[i,j]` by `1` in `x`
  4. `x[i]=Matrix(..)` replaces the row `x[i]` by the row `Matrix(..)` in `x` if the lengths of the row `x[i]` and the row `Matrix(..)` are identical, otherwise raise the exception `MatrixSyntaxError`
  - `slice` similar to list slicing
  1. `x[start1:stop1:step1,start2:stop2:step2]` returns the matrix `y`, such that `y[i,j]` is the element `x[start1+i*step1,start2+j*step2]` if it exists and `start1+i*step1<stop1`, `start2+i*step2<stop2`

2. `x[start1:stop1:step1,start2:stop2:step2]=Matrix(..)` replaces the matrix `x[start1+i*step1,start2+j*step2]` by `Matrix(..)` in `x` if the number of rows (as well as columns) of `x[start1+i*step1,start2+j*step2]` are `Matrix(..)` are identical, otherwise raise the exception `MatrixSyntaxError` `
   - `__str__` returns a string denoting the matrix in **Matrix** syntax **without white space** and follows
     that: `j` outputs `1j`, `0.0` outputs `0`, `0+2j` outputs `2j`, `0+0j` outputs `0`, `5.0` outputs `5`, `5.1` out puts `5.1` and `5.10` outputs `5.10`
   - `determinant` returns the determinant of the matrix, if the matrix is not a square matrix, raise the exception `MatrixSyntaxError`
   - `inverse` returns the inverse of the matrix， if the matrix is not a square matrix or invertible, raise the exception `MatrixSyntaxError`
- Implementation of `determinant` and `inverse` for `2-by-2, 3-by-3` matrices is **mandatory** with the exception `MatrixSyntaxError` for non-square matrix
- Optional: comlete the implementation of `determinant` and `inverse` for `n-by-n` matrices with `n>=4` is optional, as you will get extra credits, (testcases 58 and 59)

Example

```

x = Matrix("[1,2,3; 4,5,6; 7,8,9]") # can have white space print(x)

[1,2,3;4,5,6;7,8,9] # the output does not contain any white space

y = Matrix("[0,1,2; 3,4,5; 6,7,8]") z = x + y # z is an matrix instance object print(z)

[1,3,5;7,9,11;13,15,17]

z = x-y print(z)

[1,1,1;1,1,1;1,1,1]

z = x * 2 print(z)

[2,4,6;8,10,12;14,16,18]

z = x * y print(z)

[24,30,36;51,66,81;78,102,126]

print(z/2)

[12,15,18;25.5,33,40.5;39,51,63]
z = x**3 #e.g., x**3 = x*x*x print(z)
[468,576,684;1062,1305,1548;1656,2034,2412]
```

```
x == y
```

False

```
x == Matrix("[1,2,3; 4,5,6;7,8,9]")
```

True

```
x.isIdentity()
```

False

```
Matrix("[1,0,0,0; 0,1,0,0; 0,0,1,0; 0,0,0,1]").isIdentity()
```

True

```
x.isSquare()
```

False

```
Matrix("[1,2,3,4; 0,1,4,0; 0,0,1,0; 0,0,0,1]").isSquare()
```

True

```
z = x[2] print(z)
```

[7,8,9]

```
x[2,1]
```

8

```
z = x[1:3:1,0:3:2] print(z)
```

[4,6;7,9]

```
x[2] = Matrix("17,18,19") print(x)
```

[1,2,3;4,5,6;17,18,19]

```
x[1,2] = 0 print(x)
```

[1,2,3;4,5,0;17,18,19] x[1:3:1,0:3:2] = Matrix("[14,16;7,9]") print(x) [1,2,3;14,5,16;7,18,9]

```
x = Matrix("[2,-2;-1,5]") x.determinant()
```

8

```
x = Matrix("[2,4;1,0]")
z= x.inverse() print(z)
```

[0,1;0.25,-0.5]

```

## Syntax of Matrix expression

```

```
prefixExp::= '+' prefixExp prefixExp | '-' prefixExp prefixExp | '*' prefixExp
prefixExp | '*' prefixExp '(' elment ')' |
             '/' prefixExp '(' elment ')'  | 'T' prefixExp  | Matrix

infixExp::= infixExp '+' infixExp | infixExp '-'  infixExp | infixExp '*' infixExp |
infixExp '*' '(' elment ')' |
           infixExp '/' '(' elment ')'  |  infixExp 'T'  | '(' infixExp ')' | Matrix

postfixExp::= postfixExp postfixExp '+' |postfixExp postfixExp  '-'  | postfixExp
postfixExp '*' |  postfixExp '(' elment ')' '*' |
            postfixExp  '(' elment ')' '/' | postfixExp 'T' | Matrix

Matrix::= "[" Rows "]" | "[" "]"
Rows::= Row | Row ";" Rows
Row::= element | element "," Row
elment::= int | float | complex
```

``` Note: `T` denotes the transposition operator and has highest priority than `+,-,*,/`. The priorities of `+,-,*,/` follow the standard priorities

Howework 3

- Implement all the requirments for Homework 2 such that matrix instance object can be created by `x =Matrix(s, mode)`
  - mode is the string "prefix" (default value), `s` is a prefix matrix expression in **prefixExp** syntax
  - mode is the string "postfix", `s` is a prefix matrix expression in **postfixExp** syntax
  - mode is the string "infix", `s` is a prefix matrix expression in **infixExp** syntax
- In any case that the input string `s` does not follows the correct syntax, raise `MatrixSyntaxError`. Note that, the input string `s` can contain white space, " "

Example

```

x = Matrix("* [1,2,3; 3,4,5] [1,2; 3,4; 5, 6]") print(x)

[22,28;40,52]

x = Matrix("* [1,2,3; 3,4,5] [1,2; 3,4; 5, 6]", "prefix") print(x)

[22,28;40,52]

x = Matrix("[1,2,3; 3,4,5] * [1,2; 3,4; 5, 6]", "infix") print(x)

[22,28;40,52]

x = Matrix("[1,2,3; 3,4,5] [1,2; 3,4; 5, 6] *", "postfix") print(x)

[22,28;40,52] ```

## Check in

- check in `Matrix.py` file into gradebot

## Optional projects

- Some students may want to practise programming skills, here we select some projects. These projects may have been implemented. But, you can still choose some and solve them using Python or Rust. These projects are **open ended** and you will get **not** any extra credit

1. Text analysis: Sample
2. Weather Analysis: Sample
3. Binary decision diagram: Sample
4. Electronic Design Automation: Sample