HW1說明文件

前言

在 matrix.py 中，定義了 matrix 的 class，用於矩陣計算。且為了方便矩陣運算，我決定將 matrix 的運算子重載。

編譯 main.py 就可以看到答案了

注意：main.py 需要與 matrix.py 在同個目錄

功能說明

在 matrix.py 中，定義了 matrix 的 class，會在 main.py 中 import

在 main.py 中宣告了 question_a, question_b, ... , question_f 這些函數

執行了 question_a, question_b, ... , question_f 就可以看到結果了

初始化 matrix

用 list 建立 matrix 的 content，content 為實際儲存元素的地方

```
def __init__(self, array=[]):
    self.content = array
```

matrix 與 str 的轉換

定義 matrix 如何轉為字串，用於輸出 matrix

```
def __str__(self):
    return str(self.content)
```

定義 matrix 的相等

程式碼會依照以下規則檢查兩個 matrix 是否相等：

1. matrix 若行列數不同則回傳 False
2. matrix 若內容不同則回傳 False
3. 若以上條件皆不符合，則回傳 True

```
def __eq__(self, other):
    # 檢查兩個 matrix 的長度是否相等，若不相等則回傳 False
    if len(self.content) != len(other.content):
        return False
    if len(self.content[0]) != len(other.content[0]):
        return False

    # 檢查兩個 matrix 的內容是否相等，若不相等則回傳 False
    column_count = len(self.content)
    row_count = len(self.content[0])
    for i in range(column_count):
        for j in range(row_count):
            if self.content[i][j] != other.content[i][j]:
                return False

    # 若皆相等則回傳 True
    return True
```

定義 matrix 的相加

計算兩個 matrix 的相加，分為兩個步驟：

1. 創建與兩個 matrix 大小相同的 matrix，並將所有元素設定為 0
2. 將兩個 matrix 的所有元素相加，並回傳

```
def __add__(self, other):
    # 創建與兩個 matrix 大小相同的 matrix，並將所有元素設定為 0
    column_count = len(self.content)
    row_count = len(self.content[0])
    ret = matrix([[0 for _ in range(row_count)] for _ in range(column_count)])

    # 將兩個 matrix 的所有元素相加，並回傳
    for i in range(column_count):
        for j in range(row_count):
            ret.content[i][j] = self.content[i][j] + other.content[i][j]
    return ret
```

定義 matrix 的相減

計算兩個 matrix 的相減，分為兩個步驟：

1. 創建與兩個 matrix 大小相同的 matrix，並將所有元素設定為 0
2. 將兩個 matrix 的所有元素相減，並回傳

```
def __sub__(self, other):
    def __sub__(self, other):
```

```
        # 創建與兩個 matrix 大小相同的 matrix，並將所有元素設定為 0
        column_count = len(self.content)
        row_count = len(self.content[0])
        ret = matrix([[0 for _ in range(row_count)] for _ in range(column_count)])

        # 將兩個 matrix 的所有元素相減，並回傳
        for i in range(column_count):
            for j in range(row_count):
                ret.content[i][j] = self.content[i][j] - other.content[i][j]
        return ret
```

定義 matrix 的純量乘法

計算兩個 matrix 的純量乘法，分為兩個步驟：

1. 創建與被乘 matrix 大小相同的 matrix，並將所有元素設定為 0
2. 將 matrix 的所有元素乘以 scalar，並回傳

```
def scalar_multiply(self, scalar):
    # 創建與被乘 matrix 大小相同的 matrix，並將所有元素設定為 0
    column_count = len(self.content)
    row_count = len(self.content[0])
    ret = matrix([[0 for _ in range(row_count)] for _ in range(column_count)])

    # 將 matrix 的所有元素乘以 scalar，並回傳
    for i in range(column_count):
        for j in range(row_count):
            ret.content[i][j] = self.content[i][j] * scalar
    return ret
```

定義 matrix 的矩陣乘法

計算兩個 matrix 的矩陣乘法，分為兩個步驟：

1. 創建與新的 matrix，其列數與前面的matrix相同，行數與後面的matrix相同，並將每個元素設定為 0
2. 將兩個 matrix 的所有元素相乘，並回傳

```
def __mul__(self, other):
    # 創建與新的 matrix，其列數與前面的matrix相同，行數與後面的matrix相同，並將每個元素
設定為 0
    column_count = len(self.content)
    row_count = len(other.content[0])
    num_count = len(self.content[0])
    ret = matrix([[0 for _ in range(row_count)] for _ in range(column_count)])

    # 將兩個 matrix 的列行做內積，並回傳
    for i in range(column_count):
```

```
            for j in range(row_count):
                # 以下迴圈為內積
                for k in range(num_count):
                    ret.content[i][j] += self.content[i][k] * other.content[k][j]
        return ret
```

定義 matrix 的轉置矩陣

計算 matrix 的轉置矩陣，分為兩個步驟：

1. 設定轉置矩陣的長度，其列數與被轉置矩陣的行數相同，行數與被轉置矩陣的列數相同，並將每個元素設定為 0
2. 將原矩陣第 i 列第 j 行的元素設定為回傳 matrix 的第 j 列第 i 行，並回傳

```
def transpose(self):
    # 設定轉置矩陣的長度，其列數與被轉置矩陣的行數相同，行數與被轉置矩陣的列數相同，並將每個元素設定為 0
    column_count = len(self.content)
    row_count = len(self.content[0])
    ret = matrix([[0 for _ in range(column_count)] for _ in range(row_count)])

    # 將原矩陣第 i 列第 j 行的元素設定為回傳 matrix 的第 j 列第 i 行，並回傳
    for i in range(column_count):
        for j in range(row_count):
            ret.content[j][i] = self.content[i][j]
    return ret
```

定義 matrix 的反矩陣 (只能用於2*2矩陣)

計算 matrix 的反矩陣，分為三個步驟：

1. 算出矩陣的行列式值
2. 建立一個2*2的矩陣
3. 將原矩陣主對角線元素互換，次對角線元素乘以 -1，存到建立的2*2矩陣中
4. 判斷矩陣是否為有反矩陣(檢查矩陣的行列式值是否為 0)，如果為有反矩陣，則回傳

```
def inverse(self):
    # 算出矩陣的行列式值
    det = (
        self.content[0][0] * self.content[1][1]
        - self.content[0][1] * self.content[1][0]
    )

    # 建立一個2*2的矩陣
    column_count = len(self.content)
    row_count = len(self.content[0])
```

```
    ret = matrix([[0 for _ in range(column_count)] for _ in range(row_count)])

    # 將原矩陣主對角線元素互換，次對角線元素乘以 -1，存到建立的2*2矩陣中
    ret.content[0][0] = self.content[1][1]
    ret.content[0][1] = -self.content[0][1]
    ret.content[1][0] = -self.content[1][0]
    ret.content[1][1] = self.content[0][0]

    # 判斷矩陣是否為有反矩陣(檢查矩陣的行列式值是否為 0)，如果為有反矩陣，則回傳
    try:
        ret = ret.scalar_multiply(1 / det)
        return ret
    except ZeroDivisionError:
        print("error : No inverse")
```

判斷是否為對稱矩陣

判斷矩陣本身是否與轉置矩陣相同，若相同則為對稱矩陣

```
def is_symmetric(self):
    # 判斷矩陣本身是否與轉置矩陣相同，若相同則為對稱矩陣
    return self == self.transpose()
```

判斷是否為對角矩陣

判斷矩陣非主對角線上的元素是否全為 0，若全為 0，則為對角矩陣

```
def is_diagonal(self):
    # 判斷矩陣非主對角線上的元素是否全為 0，若全為 0，則為對角矩陣
    for i in range(len(self.content)):
        for j in range(len(self.content[0])):
            if i != j and self.content[i][j] != 0:
                return False
    return True
```

輸出結果

```
question_a:
A + 3B = [[-4, -2], [3, 4]]
C - 2B * E^T = [[7, 14.566370614359172, -8], [8, -1.806179973983887, -15]]
A^T = [[2, 3], [-2, -5]]

question_b:
```

```
M = [[-4, -6], [-6, -15]]
N = [[-4, 4], [9, -15]]
M is not equal to N

question_c:
P = [[2, 6], [-4, 0], [0, 9]]
Q = [[2, 6], [-4, 0], [0, 9]]
P is equal to Q

question_d:
inverse of A : [[1.25, -0.5], [0.75, -0.5]]
inverse of B : [[-0.5, -0.0], [-0.0, 0.3333333333333333]]

question_e:
A is diagonal : False
B is diagonal : True
F is diagonal : False
I is diagonal : True

question_f:
A is symmetric : False
B is symmetric : True
F is symmetric : False
I is symmetric : True
```