```
%pip install -q otter-grader
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 142.5/142.5 kB 5.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.6/101.6 kB 5.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.8/139.8 kB 6.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.1/118.1 kB 7.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 35.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 46.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.9/45.9 MB 15.0 MB/s eta 0:00:00
```

```
import sys

IN_COLAB = 'google.colab' in sys.modules

if IN_COLAB:
    !git clone https://github.com/porrashuang/CSE6740_CDA_HW0_Tests.git tests
    import otter
    grader = otter.Notebook()
else:
    print("Not running in Colab")
```

Task 1: Basic Matrix Elimination

```
### Purpose
* Solve a linear equation system using Gaussian elimination by generating elimination matrices.
* Understand what matrix multiplication is and how it can be used to perform row operations.
### Goal
Write a Python function that returns the elimination matrix used to transform a given square matrix into upper triangu

An elimination matrix E is an identity matrix with specific entries modified such that when it is left-multiplied with

The goal is to output the list of elimination matrices that, when applied in sequence, will transform the input matrix

Constraints:

* The input matrix is a list of lists (i.e., pure Python).
* The matrix must be square (n x n).
* You may not use any external libraries like NumPy.
* There will be no zero rows in the input matrix and only one solution exists.

Example:
```

A = [[2, 1], [4, 3]]

```
We want to get an upper Triangular
```

[2, 1], [0, 1]

```
Thus the Output:
```

[[[1, 0], [-2, 1]]]

```
Explaination:
We perform the operation R2 = R2 - 2 * R1, which is represented by the elimination matrix.

Note that the output is a list of n x n matrices, where each matrix corresponds to an elimination step.
```

```
def elimination_matrices(matrix):
    dimension = len(matrix)

    ##Base case
    if (dimension == 2):
```

```
        E_matrices = [[1,0],[- matrix[1][0]/matrix[0][0]], 1]
        cornerNum = matrix[dimension - 1][dimension - 1] + matrix[0][n - 1]* E_matrices[n-1][n - 2]
    else :
```

```
grader.check("q1")
```

What is the time complexity of your solution? Please return a string that describes the time complexity of your solution.

```
time_complexity = ... # (e.g "O(n)" "O(log n)" "O(nlog n)" "O(n^2)")
```

Task 2: Matrix Inversion

Purpose

- Understand the mechanism behind matrix inversion before using libraries like NumPy.

Goal

Write a Python function that uses Gaussian-Jordan elimination to compute the inverse of a given square matrix.

Constraints:

- You may not use any external libraries like NumPy.
- The input matrix is a list of lists (i.e, pure Python).
- The matrix must be square (n x n).
- The matrix may not be invertible, if it is not invertible, return None.

```
def gauss_jordan_inverse(matrix):
    ...
```

```
grader.check("q3")
```

What is the time complexity of a matrix inversion algorithm?

```
time_complexity_matrix_inv = ... # (e.g "O(n)" "O(log n)" "O(nlog n)" "O(n^2)")
```

Task 3: Postierior Probability

Purpose

- Understand the concept of posterior probability.
- Understand how to compute posterior probability using Bayes' theorem.
- This will be useful for later tasks like EM algorithm and Naive Bayes classifier.

Goal

Write a Python function that computes the posterior probability of a given event using Bayes' theorem. You are a data scientist working for a company that sells ice cream. You have data on the sales of different flavors of ice cream, and you want to compute the posterior probability of a given flavor being sold given the sales data.

$$P(Flavor|Observation) = P(Observation|Flavor) * P(Flavor)/P(Observation)$$

Constraints:

- You can assume each feature in the observation is independent of each other, and no smoothing is needed.
- You don't need to worry about zero probabilities.

The data looks like this:

```
data = [
    {"flavor": "flavor1", "hot_day": 1, "weekend": 1, "sales": 100},
    {"flavor": "flavor2", "hot_day": 1, "weekend": 0, "sales": 200},
    ...
]
```

You are given the following observation:

```
observation = {"hot_day": 1, "weekend": 1}
```

You need to compute the posterior probability of each flavor being sold given the observation (see test cases for example).

```
output = {
    "flavor1": 0.4,
    "flavor2": 0.3,
    ...
}
```

```
data = [
    {"flavor": "chocolate", "hot_day": 1, "weekend": 1, "sales": 100},
    {"flavor": "chocolate", "hot_day": 0, "weekend": 1, "sales": 50},
    {"flavor": "vanilla", "hot_day": 1, "weekend": 0, "sales": 200},
    {"flavor": "vanilla", "hot_day": 0, "weekend": 1, "sales": 100},
    {"flavor": "strawberry", "hot_day": 0, "weekend": 1, "sales": 150},
    {"flavor": "strawberry", "hot_day": 1, "weekend": 0, "sales": 50},
    {"flavor": "strawberry", "hot_day": 1, "weekend": 1, "sales": 50},
]
def compute_posterior(data, observation):
    ...
```

```
grader.check("q5")
```

Task 4: Numpy Refresh

Purpose

- Refresh your knowledge of NumPy.
- Able to perform basic operations using NumPy.

Goal

Fill in the Python function that performs the following operations using NumPy:

单元格类型不受支持。双击即可检查/修改内容。

```
import numpy as np
def q1_array_creation(data):
    """Return a NumPy array created from `data` (a Python list/tuple)."""
    ...


def q2_basic_slicing(A):
    """Return the middle 2x2 of A (assume 4x4), with columns reversed."""
    ...


def q3_boolean_indexing(A):
    """Return the elements of A that are smaller than mean."""
    ...


def q4_axis_operations(A):
    """Suppose A is 3x3, Return the sum of each column (3,) in A minus the sum of each row (3,) in A"""
    ...
```

```
def q5_matrix_operations(A):
    """Return the transpose of A, inverse of A."""
    ...


def q6_more_matrix_operations(A):
    """Return the rank of A, SVD decomposition of A."""
    ...
```

```
grader.check("q6")
```