Quiz

Let \( \mathbf{a}_1 = [1, 0, -1], \mathbf{a}_2 = [2, 1, 0], \mathbf{a}_3 = [10, 1, 2], \mathbf{a}_4 = [0, 0, 1] \). Compute the row-matrix-by-vector product

\[
\begin{bmatrix}
\mathbf{a}_1 \\
\mathbf{a}_2 \\
\mathbf{a}_3 \\
\mathbf{a}_4
\end{bmatrix} \times [7, 6, 5]
\]

Let \( \mathbf{v}_1 = [2, 1, 0, 1], \mathbf{v}_2 = [4, -1, 0, 2], \mathbf{v}_3 = [0, 1, 0, 1] \). Compute the column-matrix-by-vector product

\[
\begin{bmatrix}
\mathbf{v}_1 \\
\mathbf{v}_2 \\
\mathbf{v}_3
\end{bmatrix} \times [2, -1, 1]
\]
Neo: What is the Matrix?
Trinity: The answer is out there, Neo, and it’s looking for you, and it will find you if you want it to. *The Matrix*, 1999
Two views of same process

Row-matrix-by-vector multiplication and column-matrix-by-vector multiplication are same!

\[
\begin{bmatrix}
  a_1 \\
  a_2 \\
  \vdots \\
  a_m
\end{bmatrix}
\begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{bmatrix}
\begin{bmatrix}
  c_1 \\
  c_2 \\
  \vdots \\
  c_m
\end{bmatrix}
\times [x_1, x_2, x_3]
\]

\[
= x_1 \begin{bmatrix}
  a_1 \\
  a_2 \\
  \vdots \\
  a_m
\end{bmatrix}
+ x_2 \begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{bmatrix}
+ x_3 \begin{bmatrix}
  c_1 \\
  c_2 \\
  \vdots \\
  c_m
\end{bmatrix}
\]

\[
= \begin{bmatrix}
  x_1 a_1 \\
  x_1 a_2 \\
  \vdots \\
  x_1 a_m
\end{bmatrix}
+ \begin{bmatrix}
  x_2 b_1 \\
  x_2 b_2 \\
  \vdots \\
  x_2 b_m
\end{bmatrix}
+ \begin{bmatrix}
  x_3 c_1 \\
  x_3 c_2 \\
  \vdots \\
  x_3 c_m
\end{bmatrix}
\]
Two views of same process

Row-matrix-by-vector multiplication and column-matrix-by-vector multiplication are same!

\begin{align*}
= & \quad x_1 \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} + x_2 \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} + x_3 \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} \\
= & \quad \begin{bmatrix} x_1a_1 \\ x_1a_2 \\ \vdots \\ x_1a_m \end{bmatrix} + \begin{bmatrix} x_2b_1 \\ x_2b_2 \\ \vdots \\ x_2b_m \end{bmatrix} + \begin{bmatrix} x_3c_1 \\ x_3c_2 \\ \vdots \\ x_3c_m \end{bmatrix} \\
= & \quad \begin{bmatrix} x_1a_1 + x_2b_1 + x_3c_1 \\ x_1a_2 + x_2b_2 + x_3c_2 \\ \vdots \\ x_1a_m + x_2b_m + x_3c_m \end{bmatrix}
\end{align*}
Two views of same process

Row-matrix-by-vector multiplication and column-matrix-by-vector multiplication are same!

\[
\begin{bmatrix}
  x_1 a_1 \\
  x_1 a_2 \\
  \vdots \\
  x_1 a_m 
\end{bmatrix}
+ \begin{bmatrix}
  x_2 b_1 \\
  x_2 b_2 \\
  \vdots \\
  x_2 b_m 
\end{bmatrix}
+ \begin{bmatrix}
  x_3 c_1 \\
  \vdots \\
  x_3 c_m 
\end{bmatrix}
= \begin{bmatrix}
  x_1 a_1 + x_2 b_1 + x_3 c_1 \\
  x_1 a_2 + x_2 b_2 + x_3 c_2 \\
  \vdots \\
  x_1 a_m + x_2 b_m + x_3 c_m 
\end{bmatrix}
\]
Two views of same process

\[
\begin{bmatrix}
  x_1 a_1 + x_2 b_1 + x_3 c_1 \\
  x_1 a_2 + x_2 b_2 + x_3 c_2 \\
  \vdots \\
  x_1 a_m + x_2 b_m + x_3 c_m 
\end{bmatrix}
= 
\begin{bmatrix}
  (a_1, b_1, c_1) \cdot (x_1, x_2, x_3) \\
  (a_2, b_2, c_2) \cdot (x_1, x_2, x_3) \\
  \vdots \\
  (a_m, b_m, c_m) \cdot (x_1, x_2, x_3) 
\end{bmatrix}
= 
\begin{bmatrix}
  [a_1, b_1, c_1] \\
  [a_2, b_2, c_2] \\
  \vdots \\
  [a_m, b_m, c_m] 
\end{bmatrix} \times [x_1, x_2, x_3]
\]
Two views of same process

We showed

\[
\begin{bmatrix}
  a_1 \\
  a_2 \\
  \vdots \\
  a_m
\end{bmatrix}
\begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{bmatrix}
\begin{bmatrix}
  c_1 \\
  c_2 \\
  \vdots \\
  c_m
\end{bmatrix}
\text{times } [x_1, x_2, x_3]
\]

\[
= \begin{bmatrix}
  [a_1, b_1, c_1] \\
  [a_2, b_2, c_2] \\
  \vdots \\
  [a_m, b_m, c_m]
\end{bmatrix}
\text{times } [x_1, x_2, x_3]
\]

so

\[
\begin{bmatrix}
  a_1 \\
  a_2 \\
  \vdots \\
  a_m
\end{bmatrix}
\begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{bmatrix}
\begin{bmatrix}
  c_1 \\
  c_2 \\
  \vdots \\
  c_m
\end{bmatrix}
\text{ is same as }
\begin{bmatrix}
  [a_1, b_1, c_1] \\
  [a_2, b_2, c_2] \\
  \vdots \\
  [a_m, b_m, c_m]
\end{bmatrix}
\]
Row matrix and column matrix are same

Example:
\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12
\end{bmatrix}
\]
same as
\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12
\end{bmatrix}
\]

Same underlying math’l object, different representations

- column-list representation
- row-list representation

of a MATRIX

One operation, matrix-vector multiplication, with two interpretations:

- dot-product interpretation: output vector entries are dot-products of rows with input vector
- linear-combinations interpretation: output vector is linear combination of columns where coeff’s are input vector entries

You must memorize which is which.
The Matrix

Traditional notion of a matrix: two-dimensional array.

\[
\begin{bmatrix}
1 & 2 & 3 \\
10 & 20 & 30
\end{bmatrix}
\]

- Two rows: [1, 2, 3] and [10, 20, 30].
- Three columns: [1, 10], [2, 20], and [3, 30].
- A $2 \times 3$ matrix.

For a matrix $A$, the $i,j$ element of $A$
- is the element in row $i$, column $j$
- is traditionally written $A_{i,j}$
- but we will use $A[i,j]$
List of row-lists, list of column-lists

- One obvious Python representation for a matrix: a list of row-lists:
  \[
  \begin{bmatrix}
  1 & 2 & 3 \\
  10 & 20 & 30 \\
  \end{bmatrix}
  \]
  represented by \([\[1,2,3\], [10,20,30]]\).

- Another: a list of column-lists:
  \[
  \begin{bmatrix}
  1 & 2 & 3 \\
  10 & 20 & 30 \\
  \end{bmatrix}
  \]
  represented by \([\[1,10\], [2,20], [3,30]]\).
List of row-lists, list of column-lists

Ungraded “Quiz”: Write a nested comprehension whose value is list-of-row-list representation of a $3 \times 4$ matrix all of whose elements are zero:

$$
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
$$

Hint: first write a comprehension for a typical row, then use that expression in a comprehension for the list of lists.

Answer:

```python
>>> [[0 for j in range(4)] for i in range(3)]
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```
The matrix revealed

The Matrix Revisited (excerpt) http://xkcd.com/566/

**Definition:** For finite sets $R$ and $C$, an $R \times C$ matrix over $\mathbb{F}$ is a function from $R \times C$ to $\mathbb{F}$.

<table>
<thead>
<tr>
<th></th>
<th>@</th>
<th>#</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

- $R = \{a, b\}$ and $C = \{\@, #, ?\}$.
- $R$ is set of row labels
- $C$ is set of column labels

In Python, the function is represented by a dictionary:

```
{('a', '@'): 1, ('a', '#'): 2, ('a', '?'): 3,
 ('b', '@'): 10, ('b', '#'): 20, ('b', '?'): 30}
```
Rows, columns, and entries

<table>
<thead>
<tr>
<th></th>
<th>@</th>
<th>#</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

Rows and columns are vectors, e.g.

- Row 'a' is the vector \( \text{Vec}\{\{'@', '#', '?'\}, \{ '@': 1, '#': 2, '?': 3 \}\} \)
- Column '#' is the vector \( \text{Vec}\{\{'a', 'b'\}, \{ 'a': 2, 'b': 20 \}\} \)
Dict-of-rows/dict-of-columns representations

<table>
<thead>
<tr>
<th></th>
<th>@</th>
<th>#</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

**One representation:** dictionary of rows:

{‘a’: Vec({‘#’, ’@’, ’?’}, {’@’:1, ’#’:2, ’?’:3}),  
 ’b’: Vec({‘#’, ’@’, ’?’}, {’@’:10, ’#’:20, ’?’:30})}

**Another representation:** dictionary of columns:

{’@’: Vec({’a’,’b’}, {’a’:1, ’b’:10}),  
 ’#’: Vec({’a’,’b’}, {’a’:2, ’b’:20}),  
 ’?’: Vec({’a’,’b’}, {’a’:3, ’b’:30})}
Our Python implementation

<table>
<thead>
<tr>
<th>@</th>
<th>#</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

```python
>>> M=Mat(({'a','b'}, {'@', '#', '?'}),
        {('a','@'):1, ('a','#'):2,('a','?'):3,
         ('b','@'):10, ('b','#'):20, ('b','?'):30})
```

A class with two fields:

- **D**, a pair \((R, C)\) of sets.
- **f**, a dictionary representing a function that maps pairs \((r, c)\) \(\in R \times C\) to field elements.

```python
class Mat:
    def __init__(self, labels, function):
        self.D = labels
        self.f = function
```

We will later add lots of matrix operations to this class.

**Example:** For a Mat \(M\), \(M[r, c]\) is the entry in row \(r\), column \(c\).
### Identity matrix

For any domain $D$, there is a matrix that represents the $D$-to-$D$ identity function $f(x) = x$

\[
\begin{array}{ccc}
a & b & c \\
\hline
a & 1 & 0 & 0 \\
b & 0 & 1 & 0 \\
c & 0 & 0 & 1 \\
\end{array}
\]

**Definition:** $D \times D$ identity matrix is the matrix $\mathbb{1}_D$ such that $\mathbb{1}_D[k, k] = 1$ for all $k \in D$ and zero elsewhere.

Usually we omit the subscript when $D$ is clear from the context. Often letter $I$ (for “identity”) is used instead of $\mathbb{1}$.

Mat({'a','b','c'},{'a','b','c'},{(a,a):1,(b,b):1,(c,c):1})

**Quiz:** Write procedure `identity(D)` that returns the $D \times D$ identity matrix over $\mathbb{R}$ represented as an instance of Mat.

**Answer:**

```python
def identity(D):
    return Mat((D,D), (k,k):1 for k in D)
```
Converting between representations

Converting an instance of Mat to a column-dictionary representation:

<table>
<thead>
<tr>
<th></th>
<th>@</th>
<th>#</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

\[
\text{Mat}(\{('a', 'b'), \{('a', '@'):1, ('a', '#'):2, ('a', '?'):3, ('b', '@'):10, ('b', '#'):20, ('b', '?'):30\})
\]

\[
\Rightarrow
\{ '@': \text{Vec}(\{('a', 'b'), \{('a':1, 'b':10\})},
  '#': \text{Vec}(\{('a', 'b'), \{('a':2, 'b':20\})},
  '?': \text{Vec}(\{('a', 'b'), \{('a':3, 'b':30\})}
\}

**Quiz:** Write the procedure mat2coldict(A) that, given an instance of Mat, returns the column-dictionary representation of the same matrix.

**Answer:**

```python
def mat2coldict(A):
    return {c:Vec(A.D[0],{r:A[r,c] for r in A.D[0]}) for c in A.D[1]}
```
Module matutil

We provide a module, matutil, that defines several conversion routines:

- mat2coldict(A): from a Mat to a dictionary of columns represented as Vecs
- mat2rowdict(A): from a Mat to a dictionary of rows represented as Vecs
- coldict2mat(coldict): from a dictionary of columns (or a list of columns) to a Mat
- rowdict2mat(rowdict): from a dictionary of rows (or a list of rows) to a Mat
- listlist2mat(L): from a list of list of field elements to a Mat, the inner lists turn into rows

and also:

- identity(D, one): produce a Mat representing the $D \times D$ identity matrix
The Mat class

We gave the definition of a rudimentary matrix class:

```python
class Mat:
    def __init__(self, labels, function):
        self.D = labels
        self.f = function
```

The more elaborate class definition allows for more concise vector code, e.g.

```python
>>> M['a', 'B'] = 1.0
>>> b = M*v
>>> B = M*A
>>> print(B)
```

More elaborate version of this class definition allows operator overloading for element access, matrix-vector multiplication, etc.

<table>
<thead>
<tr>
<th>operation</th>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix addition and subtraction</td>
<td>A+B and A−B</td>
</tr>
<tr>
<td>Matrix negative</td>
<td>−A</td>
</tr>
<tr>
<td>Scalar-matrix multiplication</td>
<td>alpha*A</td>
</tr>
<tr>
<td>Matrix equality test</td>
<td>A == B</td>
</tr>
<tr>
<td>Matrix transpose</td>
<td>A.transpose()</td>
</tr>
<tr>
<td>Getting a matrix entry</td>
<td>A[r,c]</td>
</tr>
<tr>
<td>Matrix-vector multiplication</td>
<td>A*v</td>
</tr>
<tr>
<td>Matrix-matrix multiplication</td>
<td>A*B</td>
</tr>
</tbody>
</table>

You will code this class starting from a template we provide.
**Using Mat**

You will write the bodies of named procedures such as `setitem(M, k, val)` and `matrix_vector_mul(M, v)` and `transpose(M)`.

In using Mats in other code, you must use operators and methods instead of named procedures, e.g.

```python
>>> M[‘a’, ‘b’] = 1.0
>>> v = M*u
>>> b_parallel = Q*Q.transpose()*b
```

instead of

```python
>>> setitem(M, (‘a’,’B’), 1.0)
>>> v = matrix_vector_mul(M, u)
>>> b_parallel =
    matrix_vector_mul(matrix_matrix_mul(Q,
                                          transpose(Q)), b)
```

In code outside the *mat* module that uses Mat, you will import just `Mat` from the *mat* module:

```
from mat import Mat
```

so named procedures will not be imported into the namespace.

---

**In short:** Use the operators `[ ]`, `+`, `*`, `-` and the method `.transpose()` when working with Mats.
Assertions in Mat

For each procedure you write, we will provide the stub of the procedure, e.g. for `matrix_vector_mul(M, v)`, we provide the stub

```python
def matrix_vector_mul(M, v):
    """Returns the product of matrix M and vector v"
    assert M.D[1] == v.D
    pass
```

You are supposed to replace the pass statement with code for the procedure.

The first line in the body is a documentation string.

The second line is an assertion. It asserts that the second element of the pair `M.D`, the set of column-labels of `M`, must be equal to the domain of the vector `v`. If the procedure is called with arguments that violate this, Python reports an error.

The assertion is there to remind us of a rule about matrix-vector multiplication.

Please keep the assertions in your mat code while using it for this course.
Testing Mat with doctests

Because you will use Mat a lot, making sure your implementation is correct will save you from lots of pain later.

Akin to Vec, we have provided doctests

```python
def getitem(M, k):
    """
    Returns value of entry k in M
    >>> M = Mat(({1,3,5}, {'a'}),
               {(1,'a'):4, (5,'a'): 2})
    >>> M[1,'a']
    4
    """
    pass
```

You can test using copy-paste:

```python
>>> from vec import Mat
>>> M = Mat(({1,3,5}, {'a'}), ...
        {(1,'a'):4, (5,'a'): 2})
>>> M[1,'a']
4
```

You can also run all the tests at once from the console (outside the Python interpreter) using the following command:

```
python3 -m doctest mat.py
```