What is a matrix? Traditional answer

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

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 2×3 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 (Quiz)

- 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

Quiz: Write a nested comprehension whose value is list-of-*row*-list representation of a 3×4 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.

List of row-lists, list of column-lists

Quiz: Write a nested comprehension whose value is list-of-*row*-list representation of a 3×4 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:

```
>>> [[0 for j in range(4)] for i in range(3)]
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

List of row-lists, list of column-lists (Quiz)

Quiz: Write a nested comprehension whose value is list-of-*column*-lists representation of a 3×4 matrix whose i, j element is $i - j$:

$$\begin{bmatrix} 0 & -1 & -2 & -3 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \end{bmatrix}$$

Hint: First write a comprension for column j, assuming j is bound to an integer. Then use that expression in a comprehension in which j is the control variable.

List of row-lists, list of column-lists (Quiz)

Quiz: Write a nested comprehension whose value is list-of-*column*-lists representation of a 3×4 matrix whose i, j element is $i - j$:

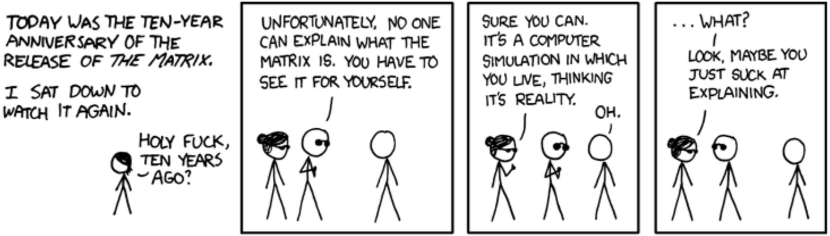
$$\begin{bmatrix} 0 & -1 & -2 & -3 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \end{bmatrix}$$

Hint: First write a comprension for column j, assuming j is bound to an integer. Then use that expression in a comprehension in which j is the control variable.

Answer:

```
>>> [[i-j for i in range(3)] for j in range(4)]
[[0, 1, 2], [-1, 0, 1], [-2, -1, 0], [-3, -2, -1]]
```

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 $\mathbb{F}$.

	@	#	?
a	1	2	3
b	10	20	30

- $R = \{\mathtt{a}, \mathtt{b}\}$ and $C = \{\mathtt{@}, \mathtt{\#}, \mathtt{?}\}$.
- 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

	@	#	?
a	1	2	3
b	10	20	30

Rows and columns are vectors, e.g.

- Row `'a'` is the vector `Vec({'@', '#', '?'}, {'@':1, '#':2, '?':3})`
- Column `'#'` is the vector `Vec({'a','b'}, {'a':2, 'b':20})`

Dict-of-rows/dict-of-columns representations

	@	#	?
a	1	2	3
b	10	20	30

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

	@	#	?
a	1	2	3
b	10	20	30

```
>>> 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.

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

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

Identity matrix

```
        a b c
      -------
 a   |  1 0 0
 b   |  0 1 0
 c   |  0 0 1
```

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 represented as an instance of `Mat`.

Identity matrix

```
       a b c
     -------
 a   |  1 0 0
 b   |  0 1 0
 c   |  0 0 1
```

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 represented as an instance of `Mat`.

Answer:

```
>>> 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:

	@	#	?
a	1	2	3
b	10	20	30

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

$$\Downarrow$$

```
{'@': Vec({'a','b'}, {'a':1, 'b':10}),
 '#': Vec({'a','b'}, {'a':2, 'b':20}),
 '?': 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.

Converting between representations

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

	@	#	?
a	1	2	3
b	10	20	30

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

$\Downarrow$

```
{'@': Vec({'a','b'}, {'a':1, 'b':10}),
 '#': Vec({'a','b'}, {'a':2, 'b':20}),
 '?': 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:

```
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 `Vec`s)
- `mat2rowdict(A)`: from a `Mat` to a dictionary of rows represented as `Vec`s
- `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)`: produce a `Mat` representing the $D \times D$ identity matrix

The `Mat` class

We gave the definition of a rudimentary matrix class:

```
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.

```
>>> 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.

operation	syntax
Matrix addition and subtraction	`A+B` and `A-B`
Matrix negative	`-A`
Scalar-matrix multiplication	`alpha*A`
Matrix equality test	`A == B`
Matrix transpose	`A.transpose()`
Getting a matrix entry	`A[r,c]`
Setting a matrix entry	`A[r,c] = alpha`
Matrix-vector multiplication	`A*v`
Vector-matrix multiplication	`v*A`
Matrix-matrix multiplication	`A*B`

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)`.

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

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

instead of

```
>>> 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 fact, in code outside the `mat` module that uses `Mat`, you will import just `Mat` from the `mat` module:

```
from mat import Mat
```

so the named procedures will not be imported into the namespace. Those named procedures in the `mat` module are intended to be used *only* inside the `mat` module itself.

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

```
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

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

We have provided a file test_mat.py with lots of examples to test against.

You can test each of these examples while running Python in interactive mode by importing Mat from the module mat and then copying the example from test_mat.py and pasting:

```
>>> 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 test_mat.py
```

This will run the tests given in test_mat.py, including importing your vec module, and will print messages about any discrepancies that arise. If your code passes the tests, nothing will be printed.

Column space and row space

One simple role for a matrix: packing together a bunch of columns or rows

Two vector spaces associated with a matrix M:
Definition:

- *column space* of M = Span {columns of M}
  Written Col M

- *row space* of M = Span {rows of M}
  Written Row M

Examples:

- Column space of $\begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{bmatrix}$ is Span $\{[1, 10], [2, 20], [3, 30]\}$.
  In this case, the span is equal to Span $\{[1, 10]\}$ since $[2, 20]$ and $[3, 30]$ are scalar multiples of $[1, 10]$.

- The row space of the same matrix is Span $\{[1, 2, 3], [10, 20, 30]\}$.
  In this case, the span is equal to Span $\{[1, 2, 3]\}$ since $[10, 20, 30]$ is a scalar multiple of $[1, 2, 3]$.

Transpose

Transpose swaps rows and columns.

```
        @   #   ?
     ----------
a   |   2   1   3
b   |  20  10  30
```

→

```
          a   b
      ------
@   |     2  20
#   |     1  10
?   |     3  30
```

Transpose (and Quiz)

Quiz: Write `transpose(M)`

Transpose (and Quiz)

Quiz: Write `transpose(M)`

Answer:

```
def transpose(M):
    return Mat((M.D[1], M.D[0]), {(q,p):v for (p,q),v in M.F.items()})
```

Matrices as vectors

Soon we study true matrix operations. But first....

A matrix can be interpreted as a vector:

- an $R \times S$ matrix is a function from $R \times S$ to $\mathbb{F}$,
- so it can be interpreted as an $R \times S$-vector:
  - *scalar-vector multiplication*
  - *vector addition*
- Our full implementation of `Mat` class will include these operations.