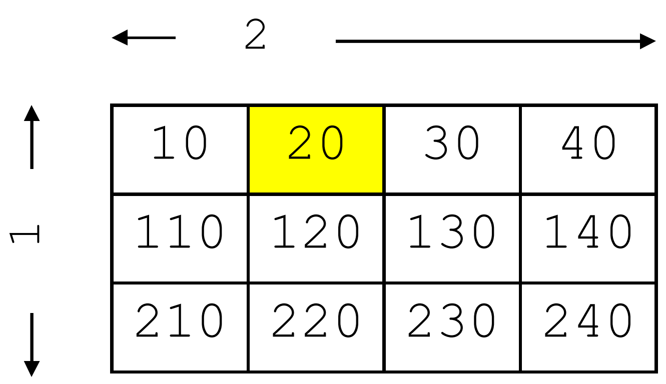
MATLAB Programming

Indexing

Can access individual elements

```
>>> block
     10    20    30    40
    110   120   130   140
    210   220   230   240
>>> block(1, 2)
     20
```

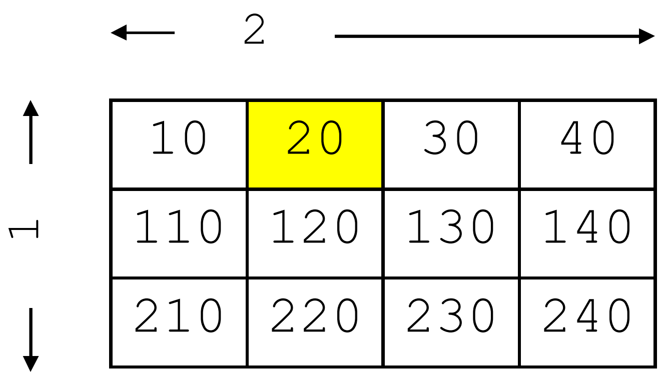


10	20	30	40
110	120	130	140
210	220	230	240

Can access individual elements

```
>>> block
     10    20    30    40
    110   120   130   140
    210   220   230   240
>>> block(1, 2)
     20
```

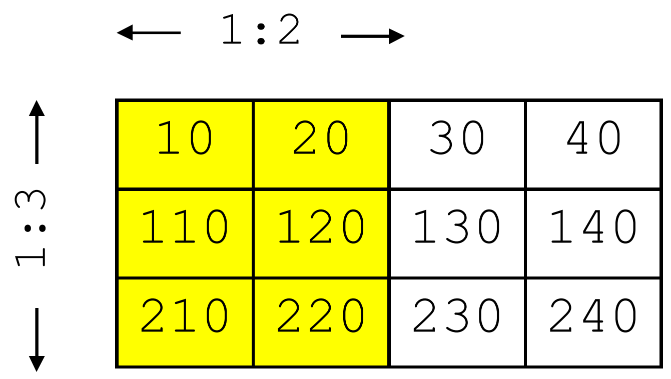
1 ï based index.



Can *slice* arrays with another list.

```
>>> 1:3
ans
    1    2    3


>>> block(1:3, 1:2)
      10     20
     110    120
     210    220
```



Why use a slice?

No loops!

Shorter…

Easier for later programmers to understand…

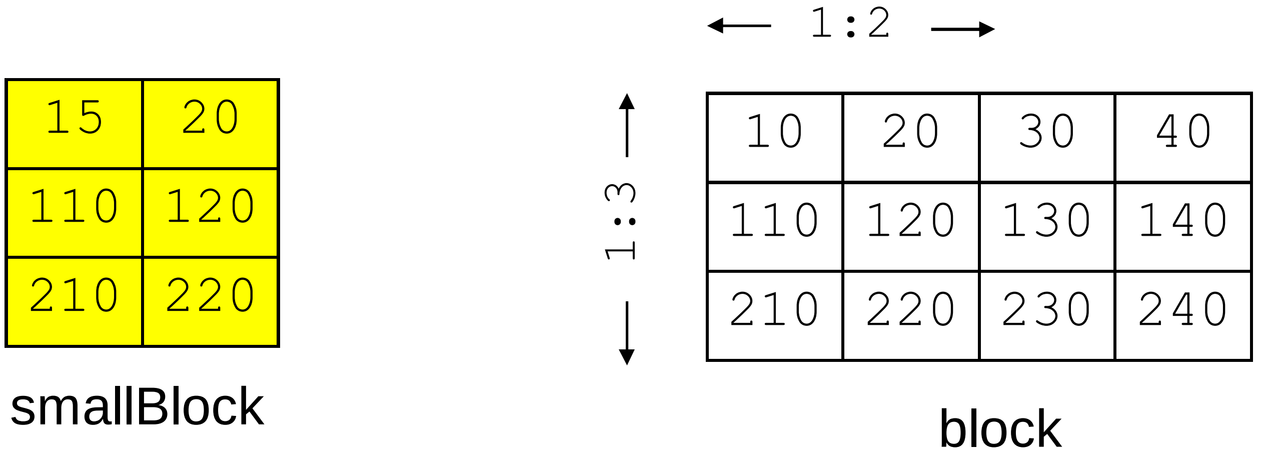
Runs faster…

Can assign to slices

```
>>> block(2, 2:3) = 0
>>> block
    10,  20,  30,  40
   110,   0,   0, 140
   210, 220, 230, 240
```

Assigning a slice makes a copy:

```
>>> smallBlock = block(1:3, 1:2);
>>> smallBlock(1,1) = 15;
>>> block(1,1)
  10
>>> smallBlock(1,1)
  15
```



Slice on both sides to shift data

```
>>> vector = [10, 20, 30, 40];
>>> vector(1:3) = vector(2:4);
>>> vector
    [20, 30, 40, 40]
```

↑

Not overwritten

Slice on both sides to shift data

```
>>> vector = [10, 20, 30, 40];
>>> vector(1:3) = vector(2:4);
>>> vector
    [20, 30, 40, 40]
```

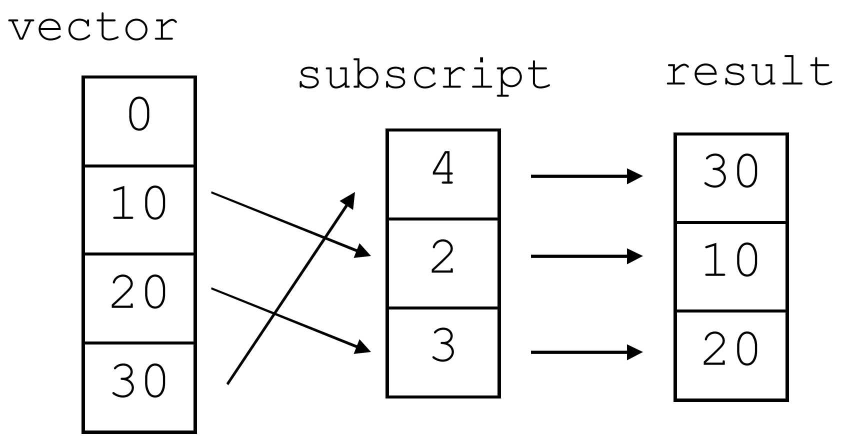
↑

Not overwritten

Investigate circshift(vector, 1) which is a MATLAB function that does something similar.

Can use lists or arrays as subscripts

```
>>> vector
    [0, 10, 20, 30]
>>> subscript = [4, 2, 3]
>>> vector( subscript )
    [30, 10, 20]
```



Comparisons

```
>>> vector
[0, 10, 20, 30]
>>> vector < 25
[ 1 1 1 0]
```

What type is the answer?

Masks:

```
>>> vector(vector < 25)
    [ 0 10 20]
```

Å Most MATLAB arrays are made of double precision floating point numbers.

Å Comparisons are arrays of booleans.

Å MATLAB displays booleans as either 1 or 0.

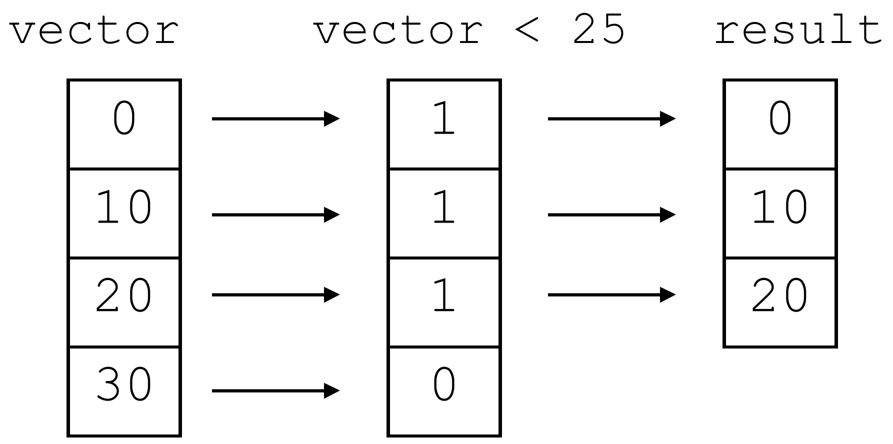
```
>>> v = [5 1 4 3];
>>> m = v < 4
  [0  1  0  1]
>>> m2 = [0 1 0 1];
>>> v(m); This is okay.
>>> v(m2); This is an error
```

These are booleans that look like doubles.

These are doubles.

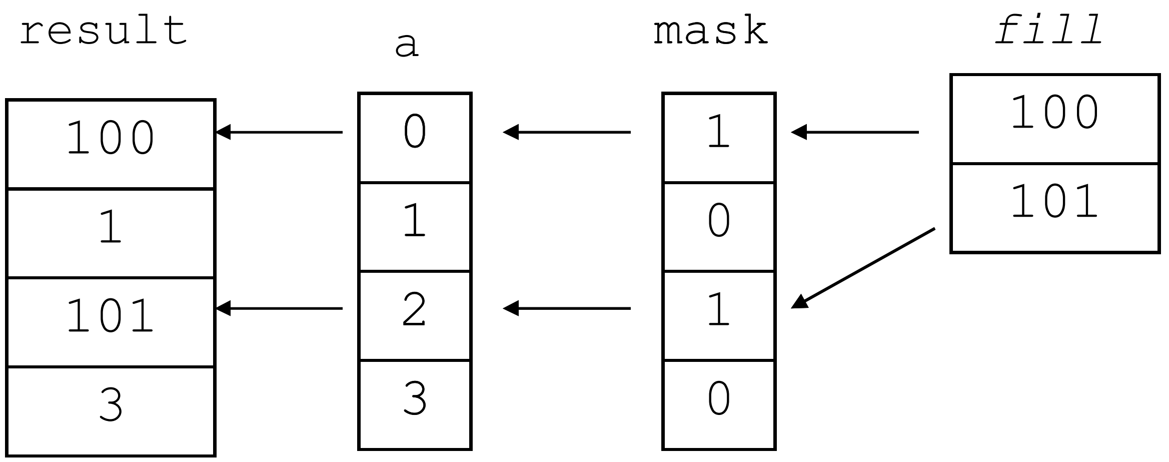
Use a Boolean subscript as a *mask*

```
>>> vector
    [0, 10, 20, 30]
>>> vector( vector < 25 )
    [0, 10, 20]
```



Use masking for assignment

```
>>> a = [0, 1, 2, 3];
>>> mask = [true, false, true, false];
>>> a(mask) = [100, 101];
>>> a
    [100, 1, 101, 3]
```



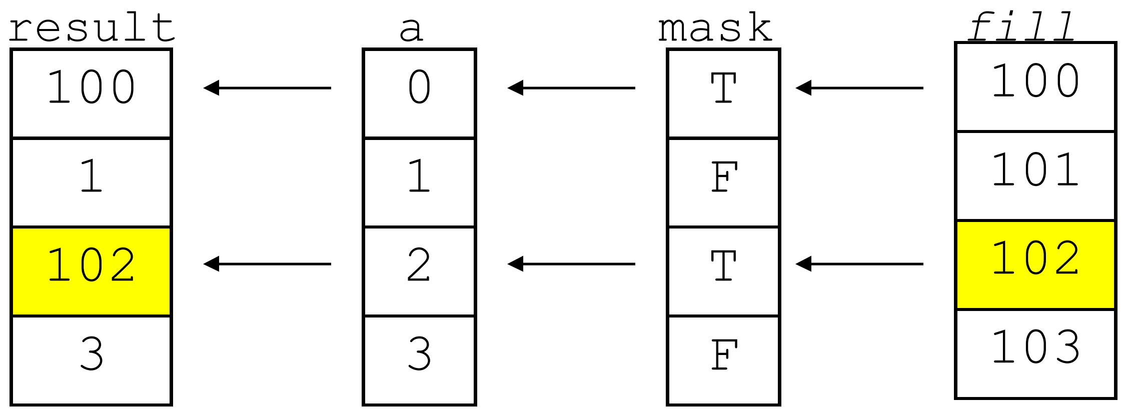
Taken

in

order

Use the mask on both sides to selectively combine two arrays:

```
>>> a = [0, 1, 2, 3];
>>> mask = [true, false, true, false];
>>> fill = [100 101 102 103];
>>> a(mask) = fill(mask)
 [100, 1, 102, 3]
```



Taken

where

True

When an array is masked, its size changes.

How can we keep the array the same size?

```
>>> v = [0 1 2 3];
>>> m = v > 1;
>>> v(m)
  [ 2 3]
>>> v .* m
  [ 0 0 2 3]
```

Booleans act like 0 and 1 in arithmetic expressions.

Logical operators &, |, and ~ operate element-wise on MATLAB arrays.

MATLAB also defined && and ||, but they operate on scalars only.

Use & and | rather than && and || to avoid confusion.

~ is ñotò ~0 is 1 and ~a for anything else is 0.

Logical operators act on all numerical types:

0 is ófalseô

Everything else is ótrueô

Be careful about relying on logical comparisons to find zeros in floating point numbers:

A number that is very small but nonzero is still ótrueô

Review:

ï Arrays can be sliced

ï Or subscripted with vectors of indices

ï Or masked with conditionals

Review:

ï Arrays can be sliced

ï Or subscripted with vectors of indices

ï Or masked with conditionals

~~LOOPS~~

created by

Richard T. Guy

February 2011