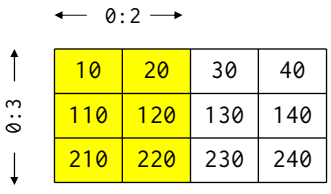
Matrix Programming

Indexing

Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.

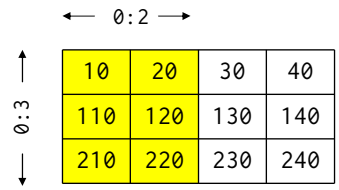
Can *slice* arrays like lists or strings

```
>>> block
array([[ 10,  20,  30,  40],
       [110, 120, 130, 140],
       [210, 220, 230, 240]])
>>> block[0:3, 0:2]
array([[ 10,  20],
       [110, 120],
       [210, 220]])
```



Can *slice* arrays like lists or strings

```
>>> block
array([[ 10,  20,  30,  40],
       [110, 120, 130, 140],
       [210, 220, 230, 240]])
>>> block[0:3, 0:2]
array([[ 10,  20],
       [110, 120],
       [210, 220]])
```



Are slices aliases or copies?

Can assign to slices

```
>>> block[1, 1:3] = 0
>>> block
array([[ 10,  20,  30,  40],
       [110,   0,   0, 140],
       [210, 220, 230, 240]])
```

Slice on both sides to shift data

```
>>> vector = array([10, 20, 30, 40])
>>> vector[0:3] = vector[1:4]
>>> vector
array([20, 30, 40, 40])
```

Not overwritten

Slice on both sides to shift data

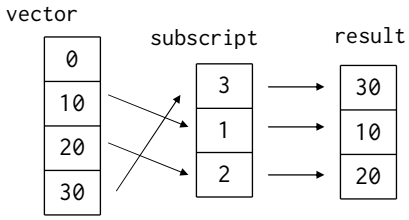
```
>>> vector = array([10, 20, 30, 40])
>>> vector[0:3] = vector[1:4]
>>> vector
array([20, 30, 40, 40])
```

Not overwritten

Is this easier to understand than a loop?

Can use lists or arrays as subscripts

```
>>> vector
array([0, 10, 20, 30])
>>> subscript = [3, 1, 2]
>>> vector[ subscript ]
array([30, 10, 20])
```



Also works in multiple dimensions

Though operation may not be obvious

```
>>> square = numpy.array([[5, 6], [7, 8]])
>>> square[ [1] ]
array([[7, 8]])
```

Also works in multiple dimensions

Though operation may not be obvious

```
>>> square = numpy.array([[5, 6], [7, 8]])
>>> square[ [1] ]
array([[7, 8]])
```

← Did we mention NumPy's excellent documentation?

Also works in multiple dimensions

Though operation may not be obvious

```
>>> square = numpy.array([[5, 6], [7, 8]])
>>> square[ [1] ]
array([[7, 8]])
```

~~LOOPS~~

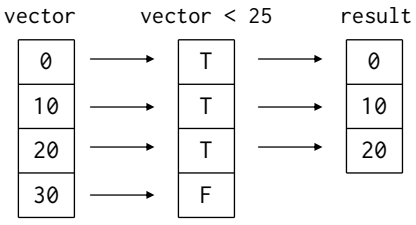
Comparisons

```
>>> vector
array([0, 10, 20, 30])
>>> vector < 25
array([ True, True, False, False ],
      dtype=bool)
```

← Data type is Boolean

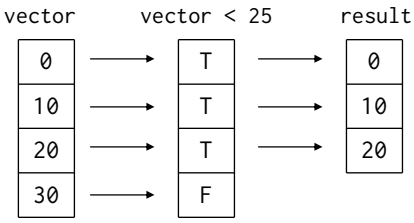
Use a Boolean subscript as a *mask*

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



Use a Boolean subscript as a *mask*

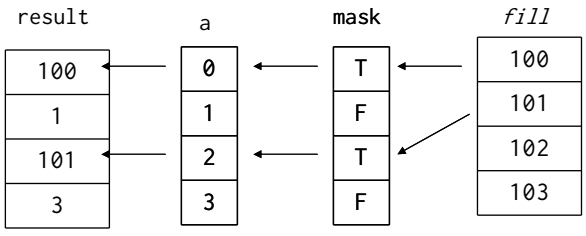
```
>>> vector
array([0, 10, 20, 30])
>>> vector[ vector < 25 ]
array([0, 10, 20])
```



Copy or alias?

Use masking for assignment

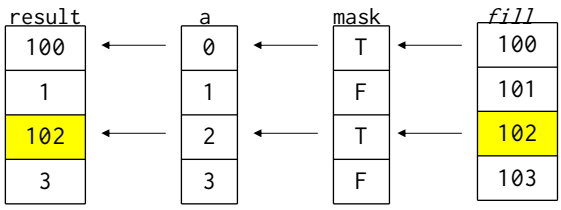
```
>>> a = array([0, 1, 2, 3])
>>> mask = array([True, False, True, False])
>>> a[mask] = array([100, 101, 102, 103])
>>> a
array([100, 1, 101, 3])
```



Taken in order

The putmask function works slightly differently

```
>>> a = array([0, 1, 2, 3])
>>> putmask(a, mask,
            array([100, 101, 102, 103]))
>>> a
array([100, 1, 102, 3])
```



Taken where True

Python does not allow objects to re-define the meaning of and/or/not

```
>>> vector = array([0, 10, 20, 30])
>>> vector <= 20
array([True, True, True, False], dtype=bool)
>>> (vector <= 20) and (vector >= 20)
ValueError: The truth value of an array with
more than one element is ambiguous.
```

software carpentry

Use `logical_and` / `logical_or` functions

```
>>> logical_and(vector <= 20, vector >= 20)
array([False, False, True, False], dtype=bool)
```

Or `|` for `or`, `&` for and

```
>>> (vector <= 20) & (vector >= 20)
array([False, False, True, False], dtype=bool)
```


software carpentry

The operators | and & deserve another look:

```
>>> a = n.array([ 1,  2 ])
>>> b = n.array([ 1, -1 ])
>>> a | b
array([ 1, -1 ])
>>> a & b
array([ 1, 2 ])
```

`logical_and`/`logical_or` treat nonzero as `True`

Use `where` instead of `if/else`

```
>>> vector = array([10, 20, 30, 40])
>>> where(vector < 25, vector, 0)
array([10, 20, 0, 0])
>>> where(vector > 25, vector/10, vector)
array([10, 20, 3, 4])
```


Use `where` instead of `if/else`

```
>>> vector = array([10, 20, 30, 40])
>>> where(vector < 25, vector, 0)
array([10, 20, 0, 0])
>>> where(vector > 25, vector/10, vector)
array([10, 20, 3, 4])
```

What do `choose` and `select` do?

Review:

- Arrays can be sliced
- Or subscripted with vectors of indices
- Or masked with conditionals

Review:

- Arrays can be sliced
- Or subscripted with vectors of indices
- Or masked with conditionals

~~LOOPS~~

created by

Richard T. Guy

November 2010

Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.