Numpy

(num-pie, not num-pee)

```
import numpy as np
```

Numpy arrays

- Numpy is a Python language extension that provides a new object -- the **array** -- for heterogeneous datasets
- Arrays are collections of objects with the same type, different from Python lists
- Operations on arrays are fast and vectorized

Numpy arrays

- To create an array from scratch, pass a list in to the `np.array` function

Numpy arrays

- Arrays can be multidimensional
- Some useful attributes to introspect:
  - **ndim**: number of dimensions
  - **shape**: number of elements along each dim
  - **size**: total number of elements

Numpy arrays

- Array operations act element-wise
- **Try these with lists and arrays to compare:**
  - multiply a list/array by 5
  - add two lists/arrays
  - subtract two lists/arrays

Numpy arrays

- Array operations are fast
- Creating a list of squared numbers from 0 to 10000 in pure-Python and with numpy:
  - `[x**2 for x in range(10000)]`
  - `np.arange(10000)**2`
- Numpy is 70x faster

Array operations

- Can be sorted in place with the .sort() method
- .max(), .min(), .mean() are handy

Array operations

- Can reshape arrays and change dimensionality, as long as number of elements is conserved
- e.g., can reshape a 1D, 100 element list to a 2D, 10 x 10 array

Numpy functions

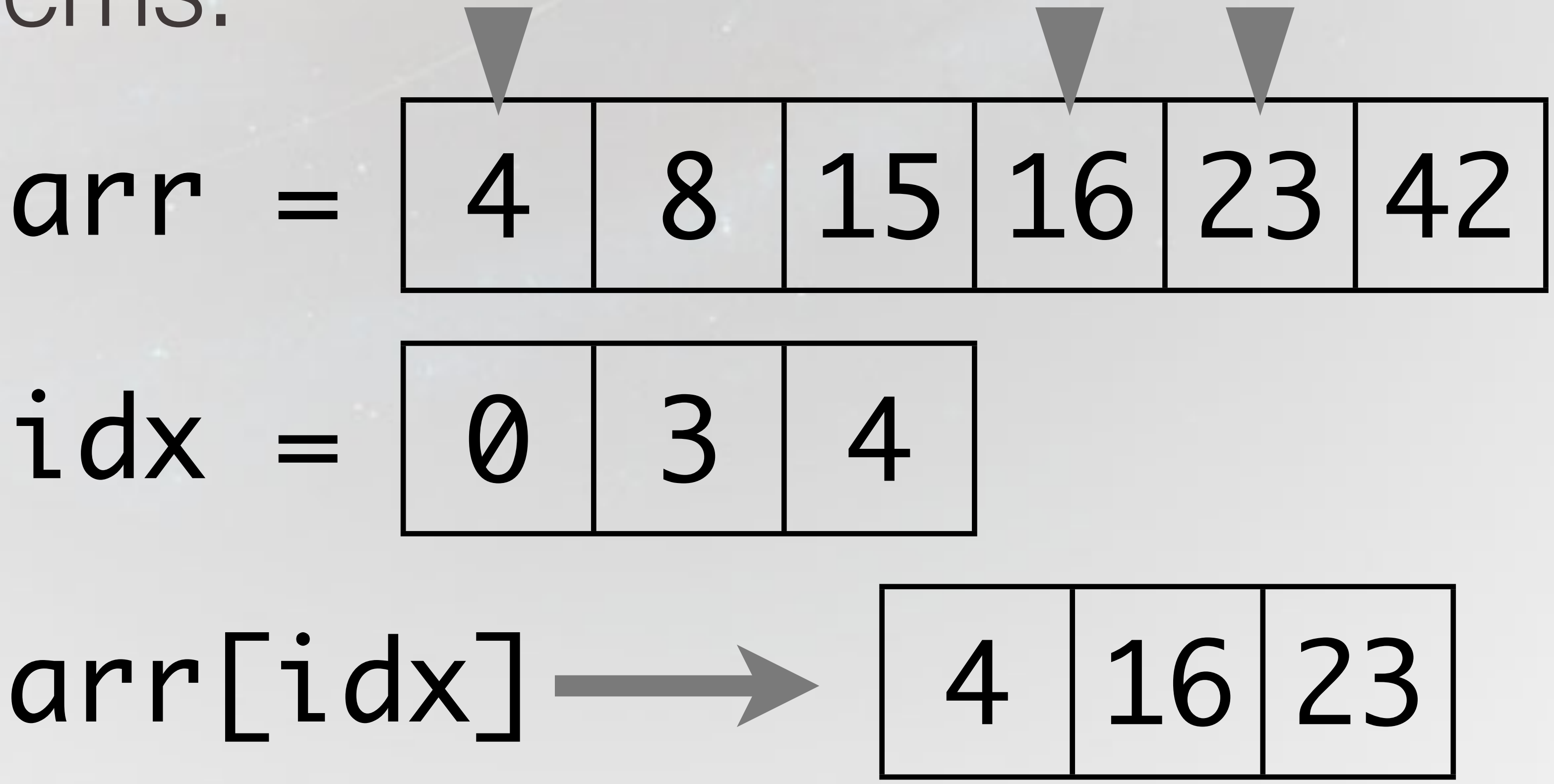
- All math functions in numpy act on arrays, and return arrays
- Other useful functions are:
    - `arange(start, stop, increment)`
    - `linspace(start, stop, number of elements)`
    - `zeros(shape), ones(shape), eye(shape)`
    - `sum(array), mean(array), median(array)`

Array indexing / slicing

- Arrays support the same slicing and indexing patterns as Python lists
- Since arrays can be multi-dimensional, can slice or index along each dimension with commas:
    - `my_array[:10,1:6,5]`

Advanced indexing

- You can also use integer numpy arrays as an index selection
- For example, let's say we have the following array, and we want to get the 0th, 3rd, and 4th items:



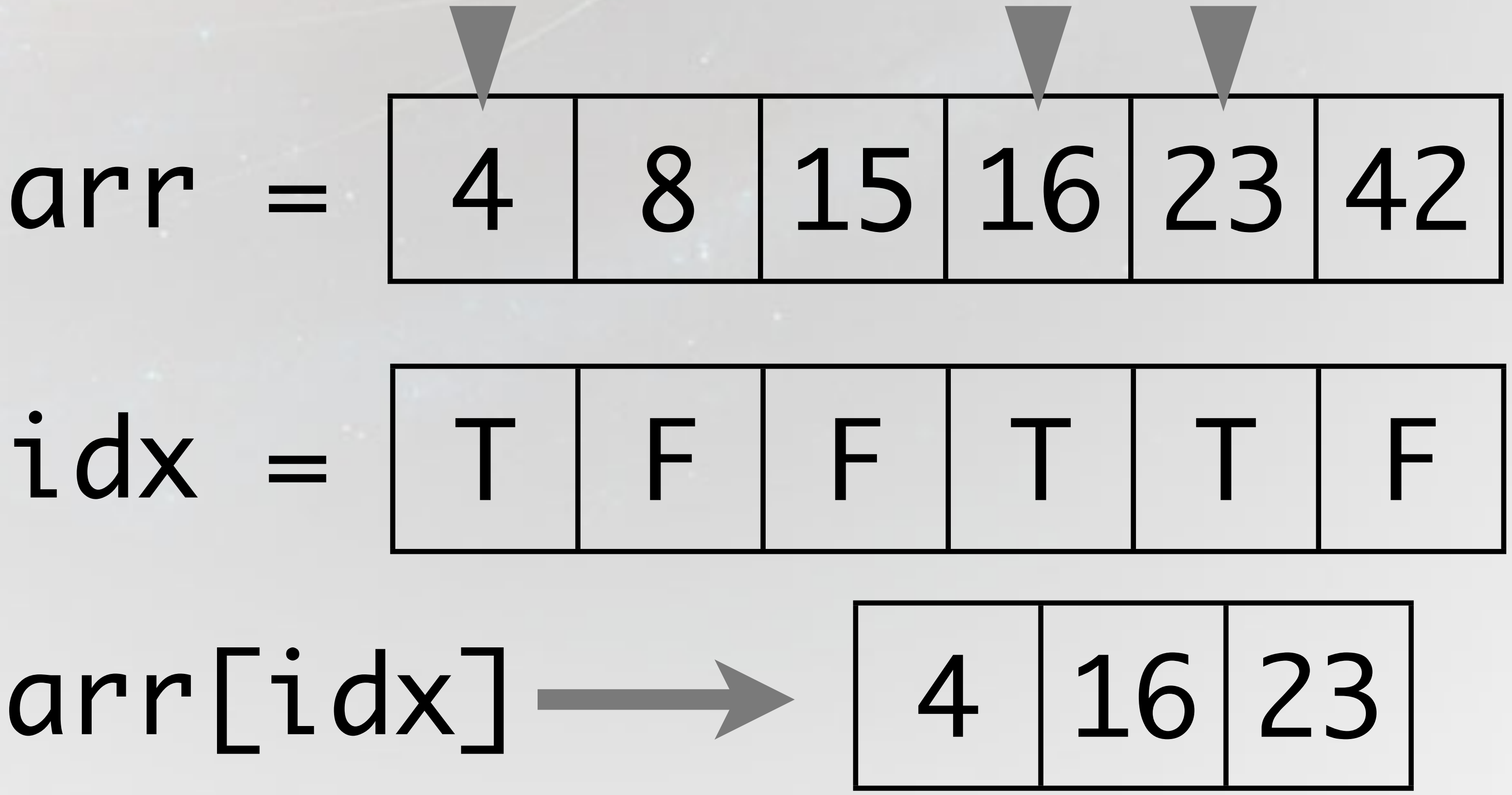
arr = 4 8 15 16 23 42

idx = 0 3 4

arr[idx] → 4 16 23

Boolean arrays

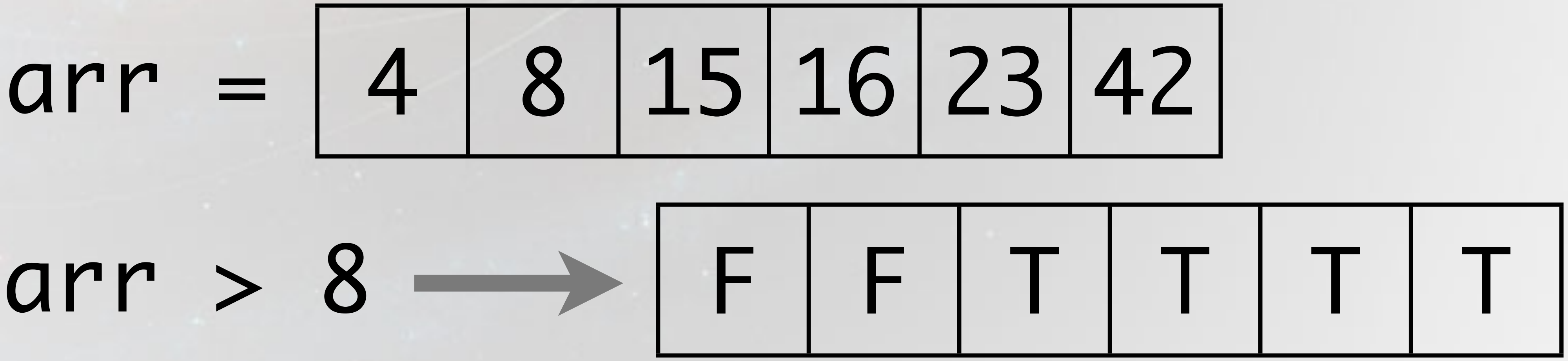
- You could do the same sub-selection of items using a **boolean array**
- A boolean array is a numpy array of `True`'s and `False`'s:



`arr =`	4	8	15	16	23	42
`idx =`	T	F	F	T	T	F
`arr[idx]` →	4	16	23			

Boolean arrays

- Boolean arrays are most useful when generated using logical operators
- For example:

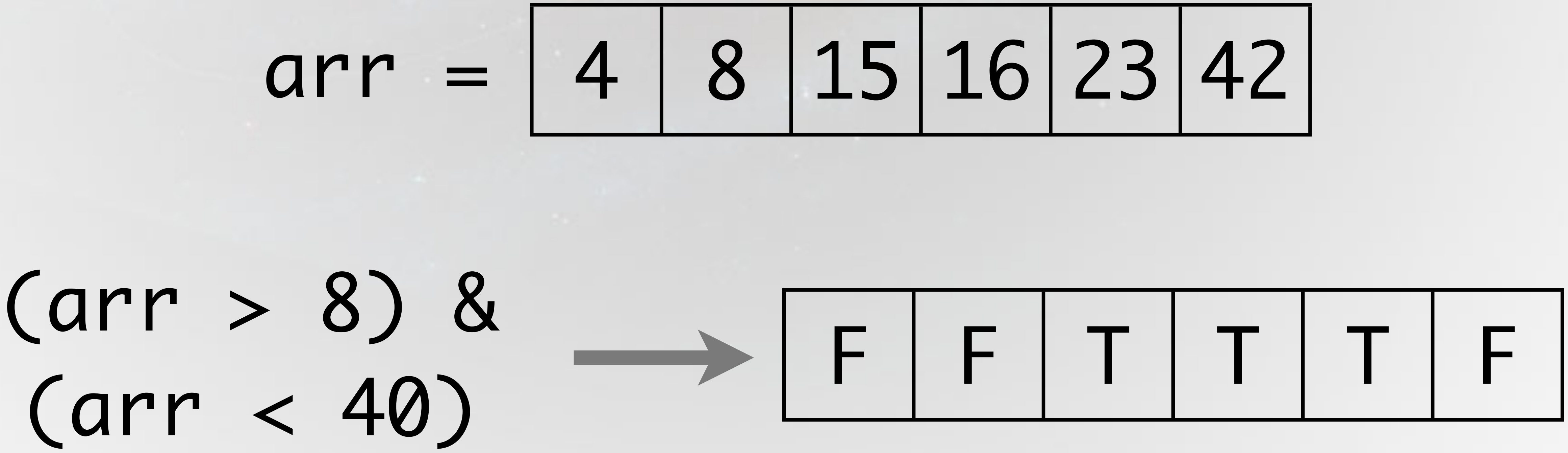


arr = | 4 | 8 | 15 | 16 | 23 | 42 |

arr > 8 → | F | F | T | T | T | T |

Boolean arrays

- We can chain together selection expressions like (arr > 8) using the bitwise operators
  - and &
  - or |



arr =

4	8	15	16	23	42

```
(arr > 8) &
 (arr < 40)
```

F	F	T	T	T	F

Boolean arrays

- Use case: let’s say I want to select out all stars within a range of RA, plot them in one color, and plot all others in another color

```
idx = (ra > 11.1324) & (ra < 31.5134)
selected = ra[idx]
not_selected = ra[np.logical_not(idx)]
```

Structured arrays

- Numpy arrays have to have homogeneous data types
- What if you have a table of numbers where each column has a type, but each is different?

Name	ID	Height	Active
“mulder”	11605	6.0	False
“scully”	42115	5.5	True

Structured arrays

- The solution is to use a Numpy structured array

```
data = [(“mulder”, 11605, 6.0, False),
        (“scully”, 42115, 5.5, True)]
arr = np.array(data, dtype=[(“Name”, str),
                            (“ID”, int),
                            (“Height”, float),
                            (“Active”, bool)])
```

Name	**ID**	**Height**	**Active**
“mulder”	11605	6.0	False
“scully”	42115	5.5	True

Structured arrays

- Indexing with the column names gets you the data for a particular column
- Using an integer gets data from a row

```
arr[“ID”] -> [11605, 42115]
arr[0] -> (“mulder”, 11605, 6.0, True)
```

Break?

Dimensionality reduction

- Sometimes you'll want to perform operations along a single axis or dimension in a numpy array
- Most functions that reduce dimensionality accept an `axis` keyword that will perform the reduction over the specified axis
- sum(arr, axis=0)

Broadcasting

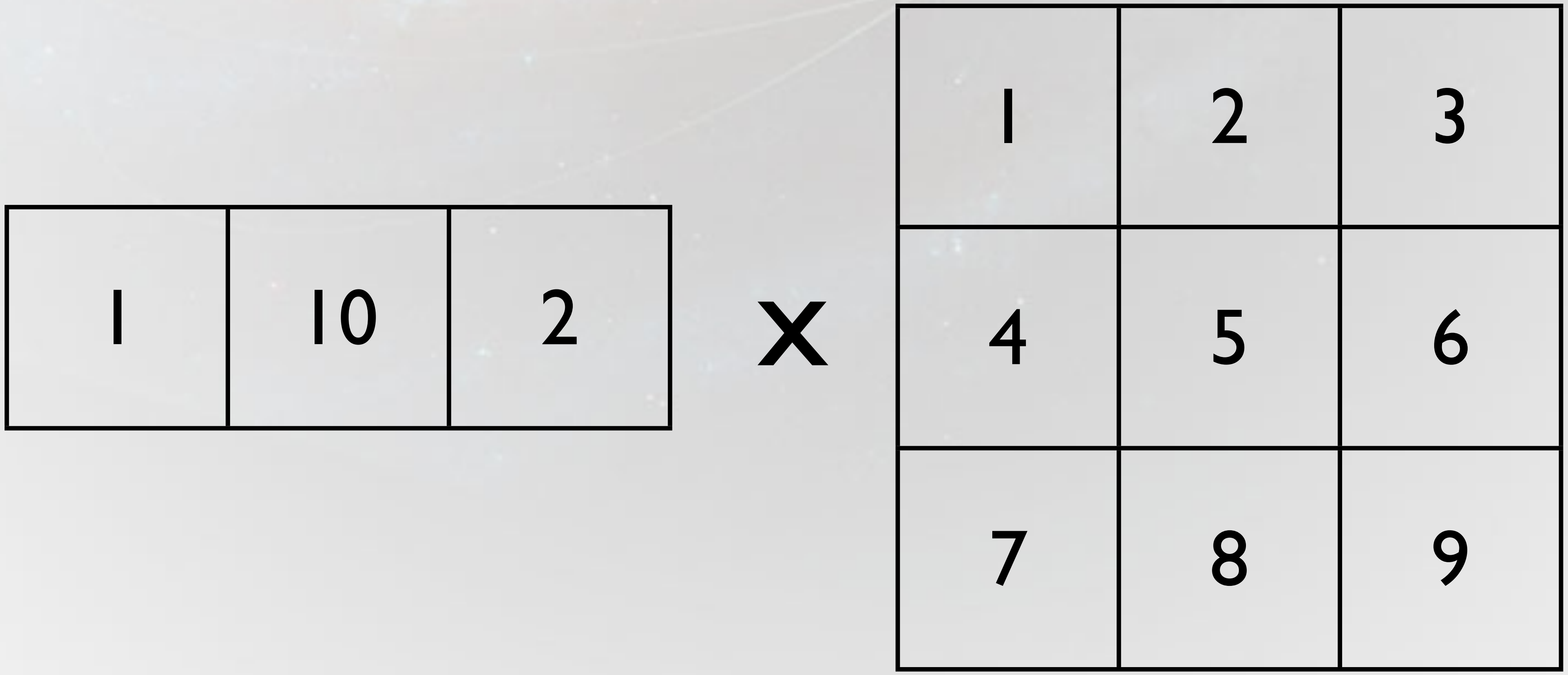
- How numpy treats arrays with different shapes during arithmetic operations

Broadcasting

- We're happy with the idea of multiplying a scalar by an array:
  - 5 * array([1,2,3,4,5]) = array(5,10,15,20,25)
- That's like multiplying a 0D array by a 1D array
- You can think of it as taking the 0D array, copying it 5 times to produce an array with the same shape as the 1D array, then performing the operation element-wise

Broadcasting

- That was a 0D array times a 1D array
- What if we add one dimension to each?
  - The rule still holds



1	10	2

x

1	2	3
4	5	6
7	8	9

Broadcasting

1	10	2
1	10	2
1	10	2

×

1	2	3
4	5	6
7	8	9

Broadcasting

1	20	6
4	50	12
7	60	18