The UK's European university

University of Kent

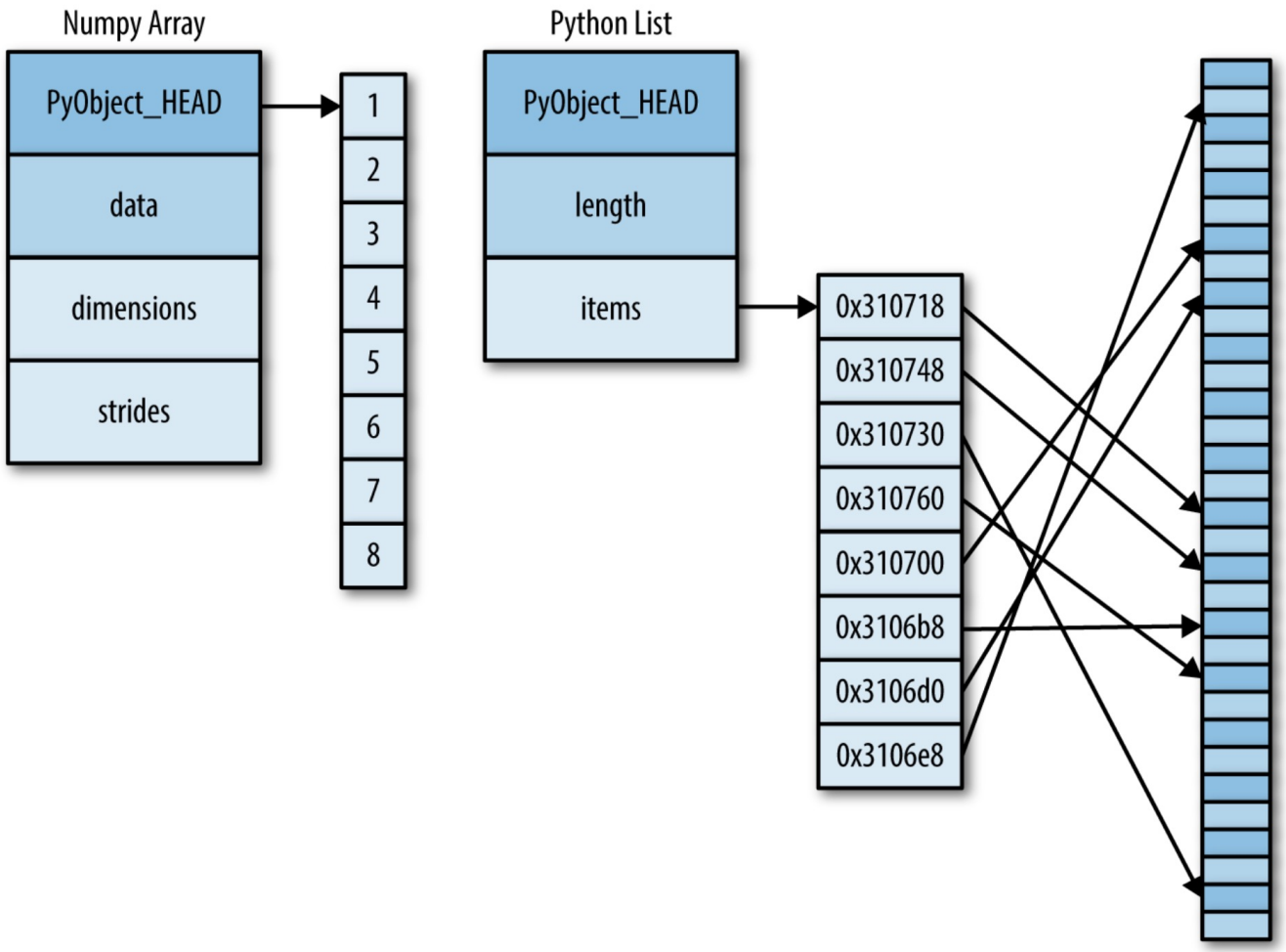
COMP8270 / PROGRAMMING FOR ARTIFICIAL INTELLIGENCE

Fernando Otero
febo@kent.ac.uk
cs.kent.ac.uk/people/staff/febo

overview:

1. More NumPy

NumPy vs Python lists:



Array slicing:

- Follows the same notation as other Python sequence objects:

`x[start:stop:step]`

- If any of these are unspecified, they default to the values **`start`**=0, **`stop`**=*size of dimension*, **`step`**=1

```
import numpy as np

x = np.arange(10)
x[:5]  # first five elements
x[::2] # every other element
x[4:7] # middle subarray
```

Multidimensional slicing:

- Work in the same way, with multiple slices separated by commas:

```
x[start:stop:step, start:stop:step]
```

dimension 1 dimension 2

```
# random int array 3x4 with values between [0, 100)
array = np.random.randint(0, 100, (3, 4))

array[:2, :3]   # first 2 rows and 3 columns
array[:3, ::2]  # all rows, every other column
array[:, 0]     # first column
```

Slicing and indexing:

```
array = np.random.randint(0, 100, (3, 4))

# access the first row, column element of the array
array[0, 0]

# this is similar to
array[0][0]
```

- In other words, if only a single value is specified in the slice, it is used as a direct index to an element
  - Use a single colon `":"` to represent the entire row or column

Boolean arrays and masks:

- NumPy provides special for boolean arrays:
  - `False` is interpreted as 0, and `True` is interpreted as 1

```
array = np.random.randint(0, 10, 10)
array < 6 # returns a boolean array


# only elements less than 6
array[array < 6]
# how many values less than 6?
np.count_nonzero(array < 6)
# how many values less than 6 in each row?
np.sum(array < 6, axis=1)
```

Boolean arrays and masks:

- You can combine operations with bitwise logic operators:
  - `&` (and), `|` (or), `^` (xor), `~` (not)
  - parenthesis are important to maintain the precedence of operators

```
array = np.random.randint(0, 10, 10)
# all values between 2 and 8
array[(array >= 2) & (array <= 8)]
# how many values between 2 and 8
np.count_nonzero([(array >= 2) & (array <= 8)])
# masks can be stored in variables
cond1 = (array >= 2)
cond2 = (array <= 8)
np.count_nonzero(cond1 & cond2)
```

Important:

- Slicing and indexing, including mask indexing, return **views** of the array
  - data can be modified!

```
array = np.random.randint(0, 10, 20)
# all values between 2 and 8 set to 0
array[(array >= 2) & (array <= 8)] = 0
# first row set to 0
array[0, :] = 0
```

Fancy Indexing (1):

- Use arrays of indices in place of single scalars

```
array = np.random.randint(0, 100, 20)

# suppose we would like to get three different values
[array[3], array[7], array[2]]

# or we can pass a single list or array of indices
array[[3, 7, 2]]
```

Fancy Indexing (2):

- The shape of the result reflects the **shape of the index array**

```
array = np.array([51, 92, 14, 71, 60, 20, 82, 86, 74, 74])


indices = np.array([[3, 7],
                    [4, 5]])


array[indices]
# array([[71, 86],
#        [60, 20]])
```

Fancy Indexing (3):

- You can combine fancy indexing with other indexing schemes:

```
array = np.random.randint(0, 10, (5, 5))

# fancy and simple indices
array[3, [0, 4]]  # first and last values of row 3

# fancy indexing with slicing
array[3:, [0, 4]] # first and last values of last 2 rows
```

Example:

- Common use of fancy indexing is the selection of subsets of values

```
# 1000 random values between 0 and 100
array = np.random.randint(0, 100, (100, 2))


# 20 random indices
indices = np.random.choice(100, 20, replace=False)


# selection of 20 random values
selection = array[indices]
```

Matrix manipulation:

- Vectors (1D) or matrices (*N*D):

add	add elements of two matrices
subtract	subtract elements of two matrices
divide	divide elements of two matrices
multiply	multiply elements of two matrices
dot	performs matrix multiplication (not element wise multiplication)
T	performs transpose of the specified matrix

File Input and Output (1):

- NumPy is able to save and load data to and from disk either in text or **binary** format

```
array = np.arange(10)
# saves to file
np.save('some_array', array)
# loads from file
np.load('some_array.npy')
```

- If the file path does not already end in `.npy`, the extension will be appended

File Input and Output (2):

- **Text** format:

```
array = np.arange(10)

# default delimiter
np.savetxt('some_array.txt', array)
# saves csv format
np.savetxt('some_array.csv', array, delimiter=",")
# loads from file
np.loadtxt('some_array.csv', delimiter=",")
```

Next lecture:

- **Pandas!**