

ENGG1811 Computing for Engineers

Week 5B:

Numpy: motivation, arrays, indexing, slicing, axes, statistical, Boolean, logic, reshape, where.

Nested for loop.

Motivating example

- The manager of a frozen food company approaches you with the following problem:
 - The company has installed 1,000 thermometers to monitor the temperature in its warehouses
 - They have collected 100,000 data points from each thermometer
 - The manager wants to know:
 - Has the temperature in any thermometer ever exceeded a threshold?
 - Which thermometers had readings exceeded the given threshold? How often did it happen?
 - Which thermometer had the highest average temperature?
- Typical modern day data processing problems:
 - Many sequences; each sequence has many data points

Illustrative example: the data

- We assume:
 - There are 5 thermometers
 - Each thermometer has 10 temperature readings
- We store the data in a list of lists
 - The variable temp_lists (see below) has 5 inner lists
 - Each inner list has 10 elements
- In general, you store
 - A data sequence in a list,
 - Multiple data sequences in a list of lists

```
temp_lists = [[ 0.3, 0.4, 0.5, 0.5, 0.1, 0.8, 0.8, 0.5, 0.0, 0.7],  
              [ 0.2, 0.4, 0.8, 0.4, 0.8, 1.8, 0.9, 0.1, 1.4, 1.7],  
              [ 1.1, 0.1, 0.8, 0.9, 0.5, 0.3, 0.2, 0.2, 1.1, 0.4],  
              [ 0.4, 0.7, 0.6, 0.6, 0.4, 0.4, 0.5, 0.0, 0.1, 0.2],  
              [ 0.2, 0.1, 0.9, 0.9, 0.3, 0.5, 0.4, 0.7, 0.2, 0.7]]
```

Thermometer 0

Thermometer 2

Illustrative example: the question

- Assuming that the threshold = 1
- For each thermometer, how many times have the readings exceeded the threshold?

```
temp_lists = [[ 0.3, 0.4, 0.5, 0.5, 0.1, 0.8, 0.8, 0.5, 0.0, 0.7],  
              [ 0.2, 0.4, 0.8, 0.4, 0.8, 1.8, 0.9, 0.1, 1.4, 1.7],  
              [ 1.1, 0.1, 0.8, 0.9, 0.5, 0.3, 0.2, 0.2, 1.1, 0.4],  
              [ 0.4, 0.7, 0.6, 0.6, 0.4, 0.4, 0.5, 0.0, 0.1, 0.2],  
              [ 0.2, 0.1, 0.9, 0.9, 0.3, 0.5, 0.4, 0.7, 0.2, 0.7]]
```

Two different solutions

- Classical programming solution
 - Nested for-loops
- Python numpy library
 - numpy is short for numerical Python

Nested for-loops

- A nested for-loop means a for-loop within a for-loop
 - You have seen nested if's before
- Example problem:
 - See illustration on the next slide for making a nested for-loop
 - Code in nested_for_prelim.py

```
temp_lists = [[ 0.3, 0.4, 0.5, 0.5, 0.1, 0.8, 0.8, 0.5, 0.0, 0.7],  
              [ 0.2, 0.4, 0.8, 0.4, 0.8, 1.8, 0.9, 0.1, 1.4, 1.7],  
              [ 1.1, 0.1, 0.8, 0.9, 0.5, 0.3, 0.2, 0.2, 1.1, 0.4],  
              [ 0.4, 0.7, 0.6, 0.6, 0.4, 0.4, 0.5, 0.0, 0.1, 0.2],  
              [ 0.2, 0.1, 0.9, 0.9, 0.3, 0.5, 0.4, 0.7, 0.2, 0.7]]
```

threshold = 1

Expected answer = [0, 3, 2, 0, 0]

Nested for-loops

For each thermometer

Find # times the readings in the thermometer > threshold

This task is a for-loop

For each reading in a thermometer

Increment count if reading > threshold

(Remark: counting heart beats)

Combine the code to get nested for-loop:

For each thermometer

For each reading in a thermometer

Increment count if reading > threshold

Nested-for loop code

nested_for_prelim.py

The following code counts the number of times that the readings in **a** thermometer has exceeded the threshold:

```
# %% Development step 2:  
count_exceeding_one_thermometer = 0  
for a_reading in readings_from_one_thermometer:  
    if a_reading > threshold:  
        count_exceeding_one_thermometer += 1
```

```
# %% Development step 3:  
counts_exceeding_all_thermometers = []  
  
for readings_from_one_thermometer in temp_lists:  
    count_exceeding_one_thermometer = 0  
    for a_reading in readings_from_one_thermometer:  
        if a_reading > threshold:  
            count_exceeding_one_thermometer += 1  
  
    counts_exceeding_all_thermometers.append(count_exceeding_one_thermometer)
```


Why numpy?

- numpy has a lot of functions that can save you time in writing code
- If we want to solve the same problem of determining the number of readings in a thermometer exceeding a threshold in numpy, the code is in `count_exceed.py`
- Line 35 uses the array of temperature and threshold to compute the answer that you want. You may not understand how to use numpy but we'd like you to appreciate that you can get the work done with merely 1 line of code

```
26 import numpy as np
27
28 # set the threshold
29 threshold = 1
30
31 # convert the list to an numpy array
32 temp_array = np.array(temp_lists)
33
34 # count the number of data points exceeding the threshold p
35 count_exceeding = np.sum(temp_array > threshold, axis = 1)
```

The numpy library

- The numpy library is a collection of functions which are very useful for data analysis, efficient computation
- Very often, a numpy function can replace many lines of code. Fewer lines of code means:
 - Shorter development time
 - Less debugging
- You combine numpy functions with your coding skills (conditional, loops etc) to solve bigger problems
- Why don't we teach you numpy from the beginning?
 - Basic coding skills give you a foundation to understand how programs are put together
 - The same reason why we don't give school children calculators before they learn their arithmetic

numpy basic concepts

- Basic concepts:
 - Creation of arrays
 - Indexing and slicing
 - Assignments
 - numpy array terminology: ndim, shape and axes

numpy arrays: creation, slicing, assignment

- Basic techniques
 - There are many ways to create numpy **arrays**
 - Enter the arrays directly
 - Convert from a list or a list of lists
 - Indexing and sliding
 - For indexing 2-d arrays, see the next slide
 - Modifying elements
- Code in `numpy_elements.py`

2-D array: shape and indexing

```
# Creating a 2-dimensional array
array2 = np.array([ [-1.2, 2, -3.1, 4.5],
                    [ 4, -5, 3.5, 7.1],
                    [ 2.7, 9, 1.7, 3.4] ])

# You can access individual elements
array2[0, 3] # 4.5
array2[1, 2] # 3.5
```

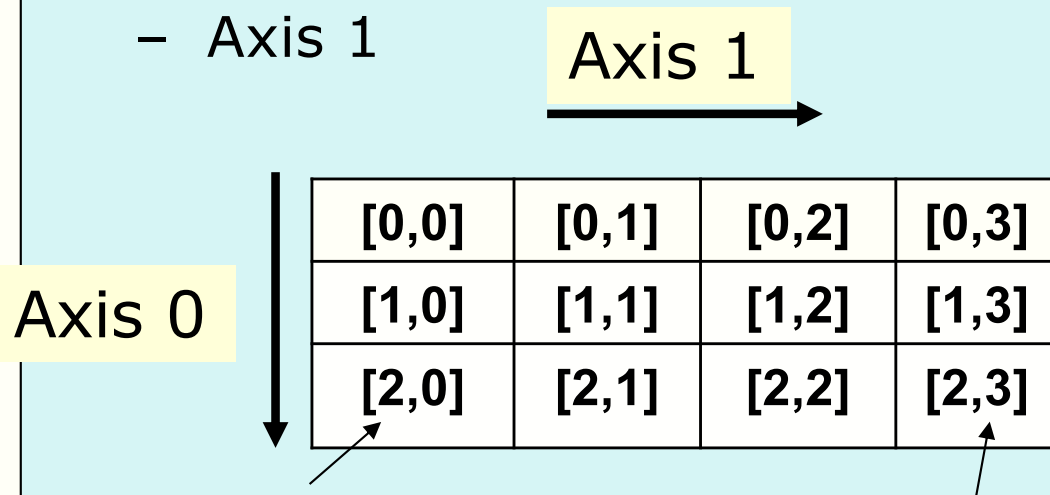
- The indexing for 2-D array is **similar to matrices** except the indices begin with zero
- We say the shape of this array is (3,4). If it were a matrix, you'd say a 3-by-4 matrix.

↓ Indices

[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]

Array axes

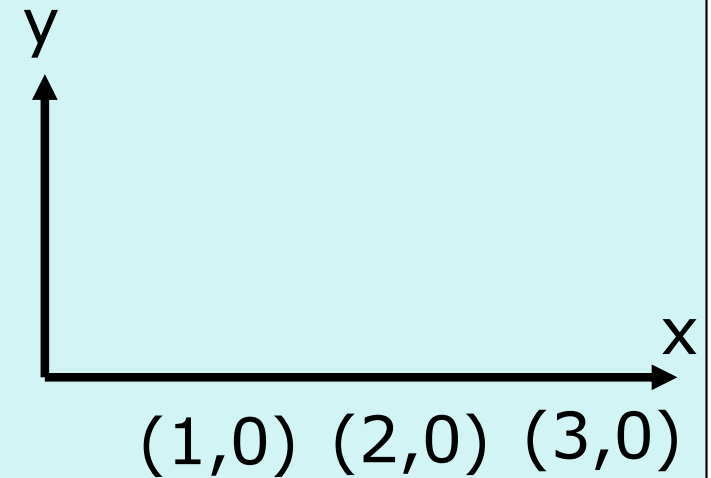
- A 2-D numpy array has 2 axes
 - Axis 0
 - Axis 1



This index is the axis 0 coordinate

This index is the axis 1 coordinate

Cartesian coordinates

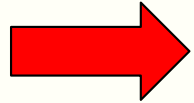


Along or parallel to x-axis, only the x-coordinates change

Rotate the x-y plane clockwise by 90°

- Along Axis 0, only Axis 0 coordinates change
- Mnemonic: *Downright*

numpy topics



Numpy functions or features	Key ideas
sum(), max(), argmax()	Summarise 2-D data arrays into 1-D array or a number Page 16 has a list of commonly used data processing functions
Boolean and logical	Elementwise operation
diff()	Difference
dtype	Data types of array elements
reshape(), ravel(), where()	Changing array shapes. Locating data with certain properties.

numpy: basic computation and statistics

- Need to prepend with numpy. or the short form that you use
- These functions can be applied to the whole array, or a particular axis
- Summarise 2-D arrays into an 1-D array or a number

numpy function	What it does
sum()	sum
mean(), median()	Mean, median
average()	Weighted average
std()	Standard deviation
max(), min()	Maximum, minimum
argmax(), argsort()	Argument that maximises / minimises
cumsum(), cumprod()	Cumulative sum and cumulative product
sort(), argsort()	sort

numpy.sum() function

Axis 1

Axis 0

Downright

- File: numpy_sum.py

- array1 is:

```
[ [-3.2,  0,  0.5,  5.8],  
  [  6, -4,  6.2,  7.1],  
  [ 3.8,  5,  2.7,  3.7]]
```

```
[ 3.1  15.3  15.2]
```

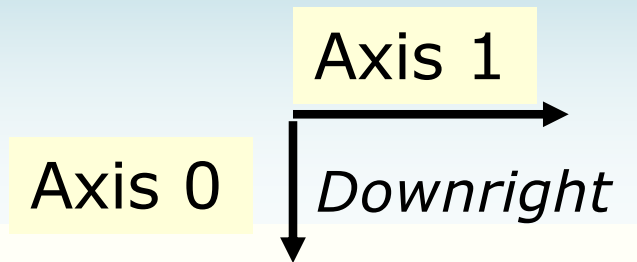
```
[ 6.6  1.  9.4 16.6]
```

```
17 print(np.sum(array1)) # default parameter value  
18                               # sum all values in the array  
19  
20 print(np.sum(array1, axis = 0)) # along axis 0  
21  
22 print(np.sum(array1, axis = 1)) # along axis 1
```

Exercise: `max()`, `argmax()`

- Go through the file `numpy_max.py`
- See whether you can figure out what `max()` and `argmax()` do
- Hint: `argmax()` has something to do with where the maximum is located
- Note: `arg` is short for argument

max() and argmax()



```
array([ [-3.2,  0,  6.2,  5.8],  
       [  6, -4,  0.5,  7.1],  
       [ 3.8,  5,  2.7,  3.7]])
```

Row index
← 0
← 1
← 2

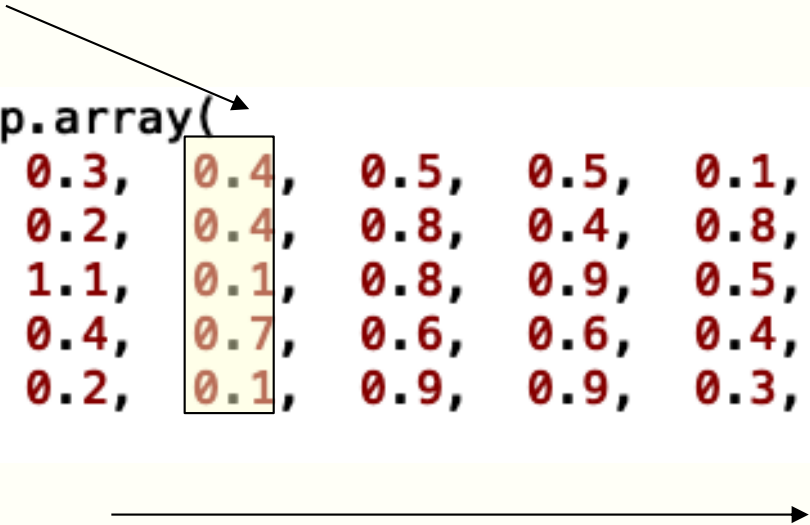
```
In [7]: np.max(array1, axis = 0)  
Out [7]: array([6. , 5. , 6.2, 7.1])
```

```
In [8]: np.argmax(array1, axis = 0)  
Out [8]: array([1, 2, 0, 1])
```

Exercise

- The file is `numpy_max_prelim.py`
- Temperature readings are stored in `temp_array`
 - Each row corresponds to a thermometer, labelled as 0,1,....,4
 - Each thermometer has 10 readings
- Your tasks
 - Find the maximum temperature in each thermometer
 - Determine which thermometer has the highest temperature at this time?

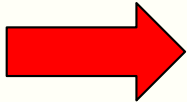
```
temp_array = np.array(  
    [[ 0.3, 0.4, 0.5, 0.5, 0.1, 0.8, 0.8, 0.5, 0.0, 0.7],  
     [ 0.2, 0.4, 0.8, 0.4, 0.8, 1.8, 0.9, 0.1, 1.4, 1.7],  
     [ 1.1, 0.1, 0.8, 0.9, 0.5, 0.3, 0.2, 0.2, 1.1, 0.4],  
     [ 0.4, 0.7, 0.6, 0.6, 0.4, 0.4, 0.5, 0.0, 0.1, 0.2],  
     [ 0.2, 0.1, 0.9, 0.9, 0.3, 0.5, 0.4, 0.7, 0.2, 0.7]]  
)
```



Time

numpy topics

Numpy functions or features	Key ideas
sum(), max(), argmax()	Summarise 2-D data arrays into 1-D array or a number Page 16 has a list of commonly used data processing functions
Boolean and logical	Elementwise operation
diff()	Difference
dtype	Data types of array elements
reshape(), ravel(), where()	Changing array shapes. Locating data with certain properties.



Boolean numpy arrays

- You can do **elementwise** comparison using `==`, `!=`, `>`, `<`, `>=`, `<=`
 - Try it out in `numpy_boolean_1_prelim.py`
 - The result is a numpy array, of the same shape, of Boolean type

```
array1 = np.array([ [-3.2, 0, 0.5, 5.8],  
                   [ 6, -4, 6.2, 7.1],  
                   [ 3.8, 5, 2.7, 3.7] ])
```

```
In [29]: array1_gt2 = array1 > 2
```

```
In [30]: array1_gt2
```

```
Out [30]:
```

```
array([[False, False,  True,  True],  
       [ True, False, False,  True],  
       [ True,  True,  True,  True]])
```

Counting the number of True's

- You can use `numpy.sum()` to count the number of True's
 - `numpy_boolean_1_prelim.py`
 - There is a quiz in the file
- We mentioned earlier the problem of determining the number of readings in a thermometer exceeding a threshold
 - The code is in `count_exceed.py`

```
26 import numpy as np
27
28 # set the threshold
29 threshold = 1
30
31 # convert the list to an numpy array
32 temp_array = np.array(temp_lists)
33
34 # count the number of data points exceeding the threshold p
35 count_exceeding = np.sum(temp_array > threshold, axis = 1)
36
```

Boolean operators

- numpy also has Boolean operators:

- The operator names are:

Python	numpy
and	&
or	
not	~

- Need parentheses around the comparisons
 - E.g. `(array1 > 3) & (array1 < 7)`
- Precedence: ~ & | (Identical to Python)
- Example (see next slide for a fuller explanation) and a quiz in `numpy_boolean_2_prelim.py`
- Forum exercise:
 - You know how to count the number of True's. How do you count the number of False's?

Illustrating **elementwise** & (and)

```
array1 = np.array([ [ 3.2, 0.5, 5.8],  
                   [ 6, -1.2, 7.1]])  
  
array1_and = (array1 > 3) & (array1 < 7)
```

array1 > 3

array1 < 7

array([[True, False, True],
 [True, False, True]])

array([[True, True, True],
 [True, True, False]])

apply logical and to corresponding elements (**elementwise**)
in the 2 arrays

array1_and

array([[True, False, True],
 [True, False, False]])

Some useful logic functions

- The full list of logic functions are at:

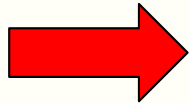
<https://numpy.org/doc/stable/reference/routines.logic.html>

numpy function	What it does
all()	True if all elements along an axis is True
any()	True if at least one element along an axis is True
allclose(), isclose()	Are elements in two arrays within a tolerance?
array_equal()	Same shape and equal elements for two arrays

The file `numpy_logic_self_study.py` has a number of examples for self-study. There are also forum exercises.

numpy topics

Numpy functions or features	Key ideas
sum(), max(), argmax()	Summarise 2-D data arrays into 1-D array or a number Page 16 has a list of commonly used data processing functions
Boolean and logical	Elementwise operation
diff()	Difference
dtype	Data types of array elements
reshape(), ravel(), where()	Changing array shapes. Locating data with certain properties.



numpy.diff()

- The code is in numpy_diff_self_study.py

```
np.array([ [-3, 0, 0, 5],  
          [ 6, -4, 6, 7],  
          [ 3, 5, 2, 3]])
```

Along axis 1

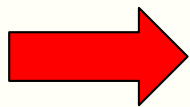
```
[[ 3  0  5]  
 [-10 10  1]  
 [ 2 -3  1]]
```

Along axis 0

```
[[ 9 -4  6  2]  
 [-3  9 -4 -4]]
```

numpy topics

Numpy functions or features	Key ideas
sum(), max(), argmax()	Summarise 2-D data arrays into 1-D array or a number Page 16 has a list of commonly used data processing functions
Boolean and logical	Elementwise operation
diff()	Difference
dtype	Data types of array elements
reshape(), ravel(), where()	Changing array shapes. Locating data with certain properties.

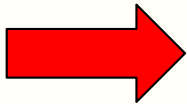


dtype

- **All** the elements in a numpy array must have the same dtype. Possible dtype's are:
 - float
 - int
 - bool
 - There are others but we won't cover them in this course
- dtype is short for datatype
- We will explore the implication of this in the file `numpy_dtype.py`

numpy topics

Numpy functions or features	Key ideas
sum(), max(), argmax()	Summarise 2-D data arrays into 1-D array or a number Page 16 has a list of commonly used data processing functions
Boolean and logical	Elementwise operation
diff()	Difference
dtype	Data types of array elements
reshape(), ravel(), where()	Changing array shapes. Locating data with certain properties.



reshape(), ravel()

- You can reshape the arrays using
 - `numpy.reshape()`
 - `numpy.ravel()`
- See examples in `numpy_reshape.py`
- Explanation on the next slide
- You will use `reshape()` to do something useful in the lab in Week 7 as well as to solve a problem in a forum exercise

ravel 1 of 2 **verb**

transitive verb

1 a : to separate or undo the texture of : **UNRAVEL**

b : to undo the intricacies of : **DISENTANGLE**

<https://www.merriam-webster.com/dictionary/ravel>


```
b = np.array([[ 3,  9,  5,  1],
              [14, 51, 16,  7],
              [12, 39, 47, 11]])
```

```
b_flat = np.ravel(b)
```

The function `numpy.ravel()` takes a "row" (default) at a time and concatenate them

```
In [65]: b_flat
```

```
Out[65]: array([ 3,  9,  5,  1, 14, 51, 16,  7, 12, 39, 47, 11])
```

```
In [9]: b_4by3 = np.reshape(b, (4,3)); print(b_4by3)
```

```
[[ 3  9  5]
 [ 1 14 51]
 [16  7 12]
 [39 47 11]]
```

numpy indexing

- We go back to the file `numpy_max.py`
-

```
30 # index_max = np.argmax(array1)
31 # indices = np.unravel_index(index_max, array1.shape)
```

- Uncomment these lines and run the file. You will find `index_max` is 7
- What is this number 7?
 - This is the “ravel” index
 - See next slide for explanation

```
array1 = np.array([ [-3.2, 0, 6.2, 5.8],  
                   [ 6, -4, 0.5, 7.1],  
                   [ 3.8, 5, 2.7, 3.7]])
```

↓ ravel()

```
In [69]: print(np.ravel(array1))  
[-3.2 0.  0.5 5.8 6. -4.  6.2 7.1 3.8 5.  2.7 3.7]
```

↑ index of
7.1 is 7

How do we get the original indices back? Let us go back to the previous slide.

numpy.where()

- The code is at numpy_where.py

↓ picture of array1 in numpy_where.py

-3	0	0	5
6	-4	6	7
3	5	2	3

array1[1,3] is 7

```
print(np.where(array1 == 7))
```

```
(array([1]), array([3]))
```

numpy.where()

- The code is at numpy_where.py

↓ picture of array1

-3	0	0	5
6	-4	6	7
3	5	2	3

```
print(np.where(array1 >= 5))
```

The indices of the elements which are ≥ 5 are:

[0 , 3]

[1 , 0]

[1 , 2]

[1 , 3]

[2 , 1]

```
(array([0, 1, 1, 1, 2]), array([3, 0, 2, 3, 1]))
```

Routines ^

Array creation routines

Array manipulation routines

Binary operations

String operations

C-Types foreign functi
numpy.ctypeslib)

Datetime support functi

Data type routines

Mathematical function
domain

Floating point error ha

Discrete Fourier Trans
numpy.fft)

Functional programmi

NumPy-specific help fi

Input and output

Linear algebra (**numpy**

Logic functions

Masked array operations

Mathematical functions

Matrix library (**numpy.matlib**)

Miscellaneous routines

Padding Arrays

Polynomials

Random sampling (**numpy.random**)

Set routines

Sorting, searching, and counting

Statistics

Test Support (**numpy.testing**)Support for testing overrides (**numpy.testing.overrides**)

Window functions

numpy has many functions

- List of numpy function categories:
 - <https://numpy.org/doc/stable/reference/routines.html>
- We will expose you to some of these functions
- Our focus:
 - Understanding
 - Use the functions for problem solving
- Exam provides numpy documentation, lecture notes etc. to reduce the need of memorization

Summary

- Data as two-dimension numpy array
- Nested for-loops
- numpy
 - Ideal for data analysis
 - Shorter code.
 - Basic concepts: Creation of arrays, indexing, slicing, axes
 - Vast collection of functions that can speed up your data analysis
 - Statistical; Boolean; Elementwise operation; reshape;
 - Operation along an axis