

# ENGG1811: Computing For Engineers

2025 Term 2

(even more on!) NumPy

Week 9: Monday 28<sup>th</sup> July, 2025

Monday 16:00 - 18:00 | TETB LG34

# Today

Housekeeping

(even more on) Numpy

Lab Tips

Housekeeping

# General Feedback on Assignment 1

- ▶ I am Very happy with everyone's work.
- ▶ If you would like more feedback, contact me via email or ask in the lab.
- ▶ Please follow the style guide as closely as possible for Assignment 2 — in particular, many people lost marks for not including function comments (see the style guide [here](#)). These should briefly provide black-box information about what inputs the function takes and what it outputs.

## General Feedback on Assignment 1 (Cont.)

- ▶ The other big style issue was not importing functions from another file! Don't copy your code—reuse the logic by calling functions.
- ▶ Methods were good, but some were a bit too technical. Methods should aim to give a high-level overview and explanation.
- ▶ When marks are released (if they aren't already), see the Collect Submission page on the assignment for specific style feedback and to view any failed tests.

## Week 10 Lab Important Info

- ▶ The Week 10 lab is conducted differently from the other labs:
  1. For the first hour, you'll do a **mock exam**.
  2. I will then go over the solutions to the exam for  $\approx 15\text{--}30$  minutes.
  3. I will then give tips for the final for  $\approx 15\text{--}30$  minutes.
  4. Outstanding marking from Week 9 (with a valid reason) will be done in the remaining time.
- ▶ There is **no MCQ** for Week 10, and you are marked *strictly* on attendance.
- ▶ Remember that **both** Assignment 2 and the second self-directed lab are due next Friday at 5 PM. Please start working on them!

# MyExperience

- ▶ You should all have (or will soon) see the window asking you to complete the "MyExperience" survey when you log into Moodle.
- ▶ Please complete the survey for this course as well as your other courses—it really does help us improve in future terms.
- ▶ Your feedback may also affect your future courses, so it can be beneficial for you too!

(even more on) Numpy



# NumPy: Shape & Size

1	2	3	4
5	6	7	8
9	10	11	12

All examples are using the top-right array

- ▶ `np.shape(array)`
  - ▶ Gives the shape — the number of rows and columns — of the given array
  - ▶ `np.shape(array) = (3, 4)`
- ▶ `np.size(array)`
  - ▶ Gives the area (number of elements) of the given array
  - ▶ `np.size(array) = 12`

## NumPy: Ravel

1	2	3	4
5	6	7	8
9	10	11	12

- ▶ `np.ravel(array)`

Using the array at the top-right

- ▶ Unravels the data into a single horizontal array (list) — converts an  $n$  dimensional array into a 1 dimensional array
- ▶ `np.ravel(array) = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]`

- ▶ **Question:** Can we achieve the same result using the reshape function? Why or why not?

## NumPy: Slicing

1	2	3	4
5	6	7	8
9	10	11	12

We can slice out a subarray using the format:

```
array[row_start:row_end:row_step,  
      column_start:column_end:column_step]
```

Here are some examples:

- ▶ `array[1:, :2] = array([[ 5, 6],  
 [ 9, 10]])`
- ▶ `array[:, :-1, :] = array([[ 9, 10, 11, 12],  
 [ 5, 6, 7, 8],  
 [ 1, 2, 3, 4]])`
- ▶ `array[:, :3, :2:] = array([[1, 2]])`

1	2	3	4
5	6	7	8
9	10	11	12

- **Recall:** When we wish to use boolean indexing, we must create an array of the same shape as the array we wish to index, and then fill each entry with either a `True` or `False` value
- For the array at the top right, we might have to make (from scratch) the following array to index it:  

```
mask = np.array([[False, False, False, False],  
                 [False, True,  True,  True],  
                 [False, True,  True,  True]])
```
- **Question:** What is the output of `array[mask]`?

## NumPy: ix\_ (Cont I)

1	2	3	4
5	6	7	8
9	10	11	12

- ▶ This is a slightly burdensome procedure — can we be a bit lazier?
  - ▶ Imagine having to create your own boolean array every time you want to do some simple indexing
- ▶ Fortunately, we can!
  - ▶ We have two options: we can select all the rows , and then boolean index on the columns **OR** we can select all the columns , and boolean index on the rows
  - ▶ **Question I:** What is the output of `array[[False, True, True], :]` ?
  - ▶ **Question II:** What is the output of `array[:, [False, True, True, True]]`?

## NumPy: ix\_ (Cont II)

1	2	3	4
5	6	7	8
9	10	11	12

### Limitation

What if we want to boolean index both the rows and columns at the same time?

- ▶ What's actually stopping us? Let's try  
`array[[False, True, True],  
 [False, True, True, True]]`

**Question:** What should this *intuitively* give us?

## NumPy: ix\_ (Cont III)

False	False	False	False	&	False	True	True	True	=	False	False	False	False
True	True	True	True		False	True	True	True		False	True	True	True
True	True	True	True		False	True	True	True		False	True	True	True

Figure: Intuitive results

- Annoyingly, we get this back instead:

### Error Message From Above

```
IndexError: shape mismatch: indexing arrays could  
not be broadcast together with shapes (2,) (3,)
```

## NumPy: ix\_ (Cont IV)

1	2	3	4
5	6	7	8
9	10	11	12

- ▶ `np.ix_(rows_boolean_array, columns_boolean_array)`
  - ▶ `rows_boolean_array` is the boolean array which selects which rows you want to keep
  - ▶ `columns_boolean_array` is the boolean array which selects which columns you want to keep
  - ▶ creates a boolean array with desired rows and columns
- ▶ Use case:

```
mask = np.ix_([False, True, True],
               [False, True, True, True])
array[mask] = np.array([[6, 7, 8],
                        [10, 11, 12]])
```

This solves our problem, and we can now be slightly lasier when it comes to boolean indexing (so long as we remember the `ix_` function!)



# NumPy: Diff

- ▶ `numpy.diff(array)`

- ▶ If we have an array  $[x_0, x_1, \dots, x_{n-1}]$  then this will return us back the list

$$[x_1 - x_0, x_2 - x_1, \dots, x_{i+1} - x_i, \dots, x_{n-1} - x_{n-2}].$$

- ▶ In plain English, it is calculating for us the (forward) difference between consecutive elements of a given array

- ▶ Example:

```
array = np.array([3, 7, 4, 9, 4, -1])  
np.diff(array) = array([ 4, -3, 5, -5, -5])
```

# NumPy: Broadcasting

- ▶ One of the many neat features of NumPy is that it allows arrays of different sizes to work together
- ▶ Intuitively, what should be the answer of adding these two arrays together?
  - ▶ Example 1:

1	2	3	4
5	6	7	8
9	10	11	12

 + 

1
2
3

 = ?

- ▶ Example 2:

1	2	3	4
5	6	7	8
9	10	11	12

 + 

1	2	3	4
---	---	---	---

 = ?

# NumPy: Broadcasting (Cont I)

Here is what NumPy is *really* doing for the above examples

► Example 1:

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline \end{array} + \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline 1 & 1 & 1 & 1 \\ \hline 2 & 2 & 2 & 2 \\ \hline 3 & 3 & 3 & 3 \\ \hline \end{array}$$

► Example 2:

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array}$$

## NumPy: Broadcasting (Cont II)

Let's do a few more examples:

► Example 3:

1	2	3	4
5	6	7	8
9	10	11	12

 $-$ 

1
---

 $=$  ?

► Example 4 (perhaps this one won't be intuitive):

1	2	3
---	---	---

 $\times$ 

1
2
3

 $=$  ?

## NumPy: Broadcasting (Cont III)

### ► Example 3:

1	2	3	4
5	6	7	8
9	10	11	12

 $-$ 

1	1	1	1
1	1	1	1
1	1	1	1

 $=$ 

0	1	2	3
4	5	6	7
8	9	10	11

### ► Example 4:

1
2
3

 $\times$ 

1	2	3
---	---	---

 $=$ 

1	1	1
2	2	2
3	3	3

 $\times$ 

1	2	3
1	2	3
1	2	3

 $=$ 

1	2	3
2	4	6
3	6	9

(3,1)      (1,3)      (3,3)      (3,3)      (3,3)

## NumPy: Broadcasting (Cont IV)

- ▶ Hopefully from these examples, we can see that broadcasting is only going to work if the arrays are **compatible** :
  1. The two arrays share a dimension of the same size and \*,
  2. One of the dimensions — for at least one of the arrays — is one
- ▶ **Examples:**
  - ▶  $2 \times 3$  and  $1 \times 3$  are compatible — they satisfy both conditions
  - ▶  $5 \times 89$  and  $5 \times 1$  are compatible — they satisfy both conditions
  - ▶  $2 \times 3$  and  $3 \times 4$  are *not* compatible — why?
  - ▶  $5 \times 6$  and  $5 \times 2$  are *not compatible* — why?
- ▶ **Question:** Why did I put an asterisk over the 'and'? What's the exception?

## NumPy: The `&`, `|` and `~` Operators

- ▶ It is a quirk of Python for `and` and `or` to behave badly with boolean conditions in NumPy — once again, we have a `workaround`
- ▶ Equivalent operations for NumPy Boolean Arrays:

English	Python	NumPy
AND	<code>and</code>	<code>&amp;</code>
OR	<code>or</code>	<code> </code>
NOT	<code>not</code>	<code>~</code>

## NumPy: The &, | and ~ Operators (cont)

1	2	3
4	5	6
7	8	9

- ▶ When using the array in the top right corner:

- ▶ `(array <= 5) & (array >= 3) \`  
`== array([[False, False, True],`  
`[True, True, False],`  
`[False, False, False]])`

- ▶ `(array >= 5) | (array <= 3) \`  
`== array([[True, True, True],`  
`[False, True, True],`  
`[True, True, True]])`

- ▶ `~(array <= 5) \`  
`== array([[True, True, True],`  
`[True, False, False],`  
`[False, False, False]])`



## NumPy: Unique

1	1	3	2
2	2	1	3
4	3	4	3

- ▶ `np.unique(array, return_counts = False)`
  - ▶ Returns an array of all the unique values of a given array
  - ▶ `return_counts = False` is an optional argument — if it is set to `True`, it will also return a list which gives back the amount of times each unique element occurs

- ▶ Example:

```
unique_vals, counts = np.unique(array,  
                                return_counts = True)  
unique_vals = np.array([1, 2, 3, 4])  
counts      = np.array([3, 3, 4, 2])
```

## Lab Tips

## Lab Tips

- ▶ You can check some of your answers on the course website (or in the starter code for Part A). Please do so before getting marked off!
- ▶ You aren't allowed to use any loops in Part A or Part B.
- ▶ Part A:
  - ▶ Think about how we discussed performing arithmetic operations efficiently in NumPy.
  - ▶ You can use NumPy functions to operate elementwise. You may want to use the square root function from some library (which one?) to do this.
- ▶ Part B:
  - ▶ Keep an eye on which methods the questions specifically ask you to use — if they do, then you have to use that method!
  - ▶ If, for Question 4 the week/Question 8 from part (b) in Week 7, you used slicing (as is wanted in this lab), then instead use the reshape approach from Week 7 (so that you've done one of each approach).

## Feedback

Feel free to provide anonymous feedback about the lab!



Feedback Form