

# ENGG1811: Computing For Engineers

2025 Term 2

While-Loops & NumPy

Week 7: Monday 14<sup>th</sup> July, 2025

Monday 16:00 - 18:00 | TETB LG34

# Today

While-Loops

NumPy

Lab Tips

# Reminders

- ▶ Virtual Lab 1 due Friday of Week 08
  - ▶ Based on Microsoft Excel.
  - ▶ Watch a series of videos in order to do the questions.
  - ▶ Get it started!
- ▶ Assignment 1 due Thursday at 5PM.
  - ▶ You should be essentially done by now , otherwise make this a top priority
  - ▶ Ask me any questions you have about style

## While-Loops

# While-Loops: Introduction

- ▶ While-loops are an alternative way to perform a loop if you find that a for-loop won't cut it
  - ▶ **For-loops:** "For every item in this list, perform this task "
  - ▶ **While-loops:** " While the following condition is true, perform this task "

# While-Loops: Advantage over For-loops

- ▶ The advantage of a while-loop is that it provides more freedom
  - ▶ You are unsure how many iterations the loop needs to go on for, but you know under what condition it should stop
    - ▶ For example, you want to keep reading input from the user until they get the correct-password
  - ▶ You want the iterator of the loop to change variably — skip items, end prematurely, or repeat the task on some specific item
    - ▶ For example, you want to update the iterator of the loop based on the remainder of the current element modulo 3

## While-Loops: 'Disadvantage' over For-loops

- ▶ With more freedom, comes the trade-off of more responsibility
  - ▶ You are now responsible for making sure the loop eventually comes to a stop
  - ▶ You are now responsible for ensuring that your iterator is updated on each iteration
- ▶ **Question:** Can all for-loops be made into a while-loop? How about vice versa?

# While-Loops: Structure

- ▶ All while-loops are essentially going to have the same structure

- ▶ Heading/Loop-Guard

- ▶ Body

- ▶ Sample:

```
while (boolean condition): # heading/loop-guard
    some sort of code      # body
```

## Questions:

- ▶ What are the two key pieces that aren't stated explicitly in this sample?
- ▶ What must the body of the loop end up doing to the boolean-condition?
- ▶ Can the boolean-condition start off as false?



# While-Loops: Examples

► Example:

```
a = 5
```

```
b = 6
```

```
while a < 10 and b > 4:
```

```
    result = a * b
```

```
    print(result)
```

```
    a += 1
```

```
    b -= 1
```

- **Questions:** What does this example do? How easy is it to replicate this into a genuine for-loop?

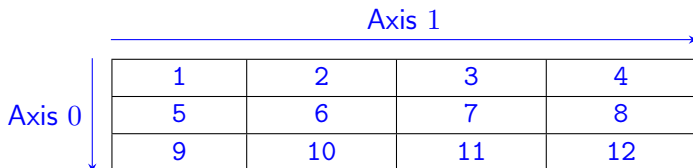
NumPy

# NumPy: Introduction

- ▶ The second half of this course is going to dedicate itself to NumPy
  - ▶ Just another library (like the `math` library)
  - ▶ Primarily used for array operations (list of lists/tables)
  - ▶ Extremely optimised — doing something in NumPy is almost always going to be quicker than doing it from scratch (sometimes significantly so)

# NumPy: Lingo

- ▶ We will use the following data for all subsequent examples:



A diagram showing a 3x4 grid of numbers. A horizontal blue arrow above the grid is labeled 'Axis 1'. A vertical blue arrow to the left of the grid is labeled 'Axis 0'.

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

- ▶ How to remember which axis is which:
  - ▶ Axis 0: "To get to ground zero, go down"
  - ▶ Axis 1: "To come 1st in a race, run across the finish line"
- ▶ The size of an `array` is denoted by `#rows × #columns`. The array above is  $3 \times 4$ .
- ▶ `array = numpy.array([[1, 2, 3, 4],  
[5, 6, 7, 8],  
[9, 10, 11, 12]])`

# NumPy: Indexing

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

- ▶ We can index and slice just as we would with a list or a list of lists:
  - ▶ `array[1][2]` gets 7
  - ▶ `array[2][3]` gets 11
- ▶ We can also use the following format, which resembles row and column coordinates :
  - ▶ `array[1, 2]` gets 7
  - ▶ `array[2, 3]` gets 11
- ▶ The second method allows for slicing both rows and columns. To see this, try comparing the following, which both attempt to get the last two rows and the middle two columns of the top right array:
  - ▶ `array[1:][1:3]`
  - ▶ `array[1:, 1:3]`

## NumPy: Indexing (Cont)

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

- ▶ We can also index multiple elements at once. For instance, using the table above, if we define:

- ▶ `rows = [0, 1, 1]`

- ▶ `columns = [0, 2, 3]`

Then, `array[rows, columns]` will yield

`np.array([1, 7, 8])`

## NumPy: Assigning

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

- ▶ Assigning values works the same way as with lists/lists of lists. The following changes the value in the second row and third column from 7 to  $-1$ :
  - ▶ `array[1, 2] = -1`
  - ▶ `array[1][2] = -1`
- ▶ However, NumPy allows us to do some operations that are not easily done with lists of lists:
  - ▶ Converting slices to a single value, e.g.,  
`array[1:3, 1:4] = -1.`
  - ▶ Replacing sub-squares with another one, such as the following for the top-left corner:  
`array[:2, :2] = np.array([[ -1, -2], [ -3, -4]]).`

## NumPy: Mean

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

- ▶ `numpy.mean(array, axis = None)`
  - ▶ Gives either the average of the entire array, each row, or each column .
- ▶ Usage:
  - ▶ `numpy.mean(array) = 6.5`
  - ▶ `numpy.mean(array, axis = 0) = [5, 6, 7, 8]`  
Note carefully that this is giving back an array, and not a single number.
  - ▶ `numpy.mean(array, axis = 1) = [2.5, 6.5, 10.5]`  
Note carefully that this is giving back an array, and not a single number.



## NumPy: Sum

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

- ▶ `numpy.sum(array, axis = None)`
  - ▶ Gives either the `sum` of the entire array, each row, or each column.
- ▶ Usage:
  - ▶ `numpy.sum(array) = 78`
  - ▶ `numpy.sum(array, axis = 0) = [15, 18, 21, 24]`  
Note carefully that this is giving back an array, and not a single number.
  - ▶ `numpy.sum(array, axis = 1) = [10, 26, 42]`  
Note carefully that this is giving back an array, and not a single number.

# NumPy: Boolean Expressions

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

- ▶ With regular python lists, if you try to naively write a boolean statement with a list, you will get an error.
- ▶ With numpy arrays, however, it will work .

```
array > 7 = array([[False, False, False, False],  
                  [False, False, False, True],  
                  [True,  True,  True,  True]])
```

# NumPy: Boolean Indexing

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

- ▶ We can index an array as if it were a list, or we can use a boolean array (known as **boolean indexing**)
- ▶ Let's say we wanted to get only the top row
  - ▶ As a list, we may index it as `array[0]` or `array[0, :]`
  - ▶ If, instead, we define

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

we can then use the above array to extract the top row with `array[indices]`.

# NumPy: Reshape

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

- ▶ `numpy.reshape(array, shape)`
  - ▶ Reshapes the given array into the provided shape, given that the array's current and new shape are 'compatible'
  - ▶ Compatible means that they have the same area/number of elements
    - ▶ The shapes  $4 \times 3$  and  $6 \times 2$  are compatible — they both equal the same area
    - ▶ The shapes  $4 \times 3$  and  $3 \times 1$  are *not* compatible — how can 12 items fit into 3 spots?

## NumPy: Reshape (Cont)

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

- ▶ Using `numpy.reshape(array, (4, 3))` gives

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

- ▶ If you know one but not both dimensions, you can write in `-1` and NumPy will work out the other dimension for you
- ▶ For example, using `numpy.reshape(array, (4, -1))` will give

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

4	8	1	12	9
---	---	---	----	---

- ▶ We are using a different array this time (check top-right).
- ▶ Sort
  - ▶ Returns the **values** from *smallest to largest*
  - ▶ `numpy.sort(array) = [1, 4, 8, 9, 12]`
- ▶ ArgSort
  - ▶ Returns the **indices** which would sort the array from *smallest to largest*
  - ▶ `numpy.argsort(array) = [2, 0, 1, 4, 3]`

## Lab Tips

# Lab Tips

- ▶ Exercise 1:
  - ▶ You need to ensure that **all logical conditions** are done **within** the loop-guard
    - ▶ No break or if statements *inside* the loop itself
- ▶ Exercise 2:
  - ▶ Remember, you are not allowed to use any loops!
  - ▶ Most of the questions can be solved in one-line of code but you don't need to force yourself to do this — it's entirely fine to use multiple lines
  - ▶ You can check your answers with the link above the set of questions
  - ▶ There is a `months` and `years` array already created for you



## Feedback

Feel free to provide anonymous feedback about the lab!



Feedback Form