# ENGG1811: Computing For Engineers

## 2025 Term 3

More on Lists & For-loops

Week 5: Monday 13th October, 2025

Monday 14:00 - 16:00 | HarpM15570

# Today

# Reminders

- ▶ By now *everyone* should have have
  begun working on Assignment 1 beginitemize
    - ▶ Due 5pm, Friday 31th October (Week 7)
    - ▶ Start it early!
    - ▶ Help sessions are available - but go early in the term as they often get busy closer to the assignment due date!
- ▶ Next week is flexi week — so no labs, no multiple choice question and you (most likely) do not have to come to uni
- ▶ Very happy with how all of you are going with the course, please keep it up (make sure to
  ask questions, receive help, attend these labs , etc)

## Assignment 1 Tips

- **Most important**: make sure to use the `test code` to check if your code is working and is giving you the correct answer!
- Even if you pass all the tests, you may not get full marks——there are `hidden tests`.
- Make sure to `read the specification *carefully*` — sometimes details are hidden away in a paragraph that are necessary to make sure your code is working as intended
- Make sure you don't copy any code or use any libraries you shouldn't.
- Remember, `style` is also marked.

# Assignment 1 Tips (Cont)

- ▶ The validity checks are only done in the final function — do not do validity checks anywhere else. Do your validity checks *first thing*; otherwise, you violate the whole point of the validity check
- ▶ Please make sure, for **all** your functions, to `return` outputs, do not(!) `print`
- ▶ Return values **exactly** as they are provided in the assignment specification — do not paraphrase, do not reword, **return exactly what we expect**
- ▶ For debugging, try using `print` to see what state your variables are in throughout the execution. This will help you test your intermediate calculations and logic.

# Lists

# Lists: Recap

- From last week, we should know:
  - Lists are just a container for our data
  - The lists have an inherent order , and lists are changeable

**Goal for Today**

Last week we mainly focused on lists being containers, but this week we will spend our time exploring what order means, and how to actually change lists .

# Lists: Order & Indexing

- We know that a list is *ordered*, but how do we actually obtain specific values?
  - How do we obtain the first value? Or the second? Or the third? Or, ... ?
  - How do we obtain the last value?
  - How do we obtain the middle value?
- We can do all of these through the *index*
  - The index records the location of a specific value within the list
- **Question:** Up until now, we have created our own lists, or used very simple lists — so why should we care about accessing some random value of a list?

# Lists: Using an Index

- The only difficulty with the index is that it starts, *unintuitively*, at $0$ rather than at $1$
- In general, to obtain the $n$th value of a list, you first write down the name of the list followed by square brackets, and then inside the square brackets $n - 1$
- Suppose we have the list `ls = [5, 7, 4, 9, 3]`
  - The $1^{\text{st}}$ value is obtained via `ls[0]` which returns 5
  - The $2^{\text{nd}}$ value is obtained via `ls[1]` which returns 7
  - The $3^{\text{rd}}$ value is obtained via `ls[2]` which returns 4
  - The $4^{\text{th}}$ value is obtained via `ls[3]` which returns 9
  - The $5^{\text{th}}$ value is obtained via `ls[4]` which returns 3

# Lists: Examples

Suppose `ls = [93, -26, 56, 16, 0, 1]`
- **Questions:**
  - What is `ls[3]`? How about `ls[1]`?
  - What should I write to get `-26`? How about `93`?
  - What is `ls[-1]`? How about `ls[-2]`?
  - What is `ls[6]`? How about `ls[-6]`?

# Lists: Slicing

- What if we wanted to obtain *multiple* values all at once, rather than just one at a time?

  - This is called **slicing**

  - We use a : to delineate between a start and end point

  - If `ls = [33, -9, 36, -15, 12]` then
    `ls[2:4] = [36, -15]`

    - The start point is 2
    - The end point is 4

  - Note carefully that although the start point is included , the

    end point is **not** included

- **Questions:** Let
  `ls = [-52, 0, -48, -94, 66, -7, 82, -15]`

  - What will `ls[0:100]` return?
  - What will `ls[0:-1]` return?
  - What will `ls[4:4]` return?

# Lists: Slicing (Cont)

- ▶ What if we want to skip some values in a slice?
    - ▶ We do slicing the normal way, but we include an additional `:` after the end point to specify the step size
    - ▶ If `ls = [-52, 0, -48, -94, 66, -7, 82, -15]` then `ls[0:8:2] = [-52, -48, 66, 82]`
- ▶ **Questions:** Once again, let `ls = [-52, 0, -48, -94, 66, -7, 82, -15]`
    - ▶ What is `ls[3:6:2]`?
    - ▶ What is `ls[0:8:-1]`?
    - ▶ What is `ls[7:0:-1]`?
    - ▶ What is `ls[::-1]`?
    - ▶ What is `ls[:len(ls):len(ls) - 1]`?
    - ▶ What is `ls[:3]`? What is `ls[3:]`?

Now that we know how to *obtain* specific values, how do we *change* specific values in a list?

# Lists: Changing Values

▶ It's as easy as `assigning` the value to the position!

▶ Suppose `ls = [93, -26, 56, 16, 0, 1]`

  ▶ Doing `ls[0] = 0` will turn `ls` into

$$ls = [0, -26, 56, 16, 0, 1]$$

  ▶ Then doing `ls[1] = 1` will turn `ls` into

$$ls = [0, 1, 56, 16, 0, 1]$$

  ▶ Then doing `ls[2] = 2` will turn `ls` into

$$ls = [0, 1, 2, 16, 0, 1]$$

  and so on ...

▶ **Question:** If we want to change *all* the values in a list, is it best to do it one by one? What should we do if we wanted to update all values in a *slice*?

# Lists: Changing Ranges

▶ To modify a range of elements in a list, we assign a new list to that slice.

▶ For example, again given `ls = [93, -26, 56, 16, 0, 1]`:

▶ Assigning `ls[1:3] = [42, 64]` will update `ls` to:

$$ls = [93, 42, 64, 16, 0, 1]$$

▶ Assigning `ls[1:3] = 2` will not work. Instead, use:

```
for i in range(1, 3):
    ls[i] = 2
```

This results in:

$$ls = [93, 2, 2, 16, 0, 1]$$

# Lists: Lists of Lists

- List of lists occur very frequently, and so we should dedicate some time going over how they work
  - A table is a list of lists
  - Matrices can be thought of as a list of lists
  - Pascal's triangle can be thought of as a list of lists
  - And many, many, more!
- **Examples:**
  - `ls = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]`
  - `ls = [[1, 2, 3], ["Uno", "Dos","Tres"]]`
  - `ls = [[0], [1, 2], [3, 4, 5], [6, 7, 8, 9]]`
  - `ls = [[56456]]`
  - `ls = [[], [], []]`

# Lists: Indexing Lists of Lists

Let `ls = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]`

- ▶ Using only one index, we have:
    - ▶ `ls[0] = [1, 0, 0]`
    - ▶ `ls[1] = [0, 1, 0]`
    - ▶ `ls[2] = [0, 0, 1]`
- ▶ Using both indices, we have:
    - ▶ `ls[0][0] = 1, ls[0][1] = 0, ls[0][2] = 0`
    - ▶ `ls[1][0] = 0, ls[1][1] = 1, ls[1][2] = 0`
    - ▶ `ls[2][0] = 0, ls[2][1] = 0, ls[2][2] = 1`
- ▶ **Questions:**
    - ▶ How do I get the bottom-right value? How about the top left? How about the centre?
    - ▶ How do I get the value in the last row and last column? How about the value in the first row and first column?
    - ▶ Play around with slicing here on your own.

# For-Loops

# For-Loops: Via Value or Via Index

▶ Now that we understand indexing, we can use it in for-loops.
  ▶ Consider the following loop:
    ```
    short_alphabet = ["a", "b", "c"]
    for letter in short_alphabet:
        print(letter)
    ```
  ▶ Instead of iterating over the values of the list, we can
    iterate over the indices of the list:
    ```
    short_alphabet = ["a", "b", "c"]
    for i in range(len(short_alphabet)):
        print(short_alphabet[i])
    ```
▶ **Questions:**
  ▶ Which loop is more flexible?
  ▶ Which loop contains more information?
  ▶ Which loop is better?

# For-Loops: Via Value or Via Index (Cont)

- Some examples of things we can achieve by using index include:
    - Skipping Elements : If I only wanted to make a change to every second element, I could do something like:
        ```
        short_alphabet = ["a", "b", "c"]

        for i in range(0, len(short_alphabet), 2):
            print(short_alphabet[i])
        ```
    - Comparing Values : We can compare two (or more) consecutive values against each other

Tips

# Tips: Lab Tips

- ► Exercise 1:
  - ► Remember you can index a range using `slicing`
- ► Exercise 2:
  - ► Remember if you index a list of lists you get the whole list not the elements within
- ► Exercise 3:
  - ► The example is different to the one in the lecture, so you can't use the exact same code
  - ► Think of the `relationship` between the threshold value and the values `before and after` the line increases past the threshold

# Feedback

Feel free to provide anonymous feedback about the lab!



Feedback Form