# Week 8

COMP10001

# What is a “library”? How do we access them?

# What is a “library”? How do we access them?

- A library contains a group of methods and/or variables that can extend Python to perform a more diverse set of tasks, without the need to write our own functions

# What is a “library”? How do we access them?

- A library contains a group of methods and/or variables that can extend Python to perform a more diverse set of tasks, without the need to write our own functions
- We import libraries:
  - `import numpy`

# What is a “library”? How do we access them?

- A library contains a group of methods and/or variables that can extend Python to perform a more diverse set of tasks, without the need to write our own functions
- We import libraries:
  - `import numpy`
- And we can assign temporary names:
  - `import numpy as np`

# What is a “defaultdict”? How do we initialise and use one?

# What is a “defaultdict”? How do we initialise and use one?

- Just like a dictionary, but new keys are initialised to a default value so that we don’t have to initialise them explicitly!

# What is a “defaultdict”? How do we initialise and use one?

- Just like a dictionary, but new keys are initialised to a default value so that we don’t have to initialise them explicitly!
- We need to import defaultdicts from the collections library:
  - `from collections import defaultdict`

# What is a “defaultdict”? How do we initialise and use one?

- Just like a dictionary, but new keys are initialised to a default value so that we don’t have to initialise them explicitly!
- We need to import defaultdicts from the collections library:
  - `from collections import defaultdict`
- And we should consider what type the default value in the defaultdict should be!
  - `tally = defaultdict(int)`

# Rewrite the following with a defaultdict

```
my_dict = {}
for i in range(10):
    if i % 3 in my_dict:
        my_dict[i % 3].append(i)
    else:
        my_dict[i % 3] = [i]
```

# Rewrite the following with a defaultdict

```
my_dict = {}
for i in range(10):
    if i % 3 in my_dict:
        my_dict[i % 3].append(i)
    else:
        my_dict[i % 3] = [i]
```

```
from collections import defaultdict

my_dict = defaultdict(list)
for i in range(10):
    my_dict[i % 3].append(i)
```

What is a “bug”? What are some debugging strategies which we can use when we find an error?

# What is a “bug”? What are some debugging strategies which we can use when we find an error?

- Bug == problem!
  - Our code isn’t working how we think it should!

# What is a “bug”? What are some debugging strategies which we can use when we find an error?

- Bug == problem!
  - Our code isn’t working how we think it should!
- Debugging strategies

# What is a “bug”? What are some debugging strategies which we can use when we find an error?

- Bug == problem!
  - Our code isn’t working how we think it should!
- Debugging strategies
  - Compare test cases to expected outputs
  - Find the section that’s relevant to the error (e.g. write test cases for your helper functions)
  - Use diagnostic print statements to check what’s happening during code execution

# What are the three types of errors we've learned about?

# What are the three types of errors we've learned about?

- Syntax
- Run-time
- Logic

# What are the three types of errors we've learned about?

- Syntax
  - Code won't run as there's something wrong with the **syntax**
- Run-time
- Logic

# What are the three types of errors we've learned about?

- Syntax
  - Code won't run as there's something wrong with the **syntax**
- Run-time
  - The syntax is fine! Code won't run as there's an error that **stops** it **during execution**
- Logic

# What are the three types of errors we've learned about?

- Syntax
  - Code won't run as there's something wrong with the **syntax**
- Run-time
  - The syntax is fine! Code won't run as there's an error that **stops** it **during execution**
- Logic
  - Code runs without appearing to create an error! But the output doesn't match what we expect!

How can we use testing to find bugs or confirm our code runs properly? What are some strategies to write comprehensive test cases?

# How can we use testing to find bugs or confirm our code runs properly? What are some strategies to write comprehensive test cases?

- To be confident our code is correct, we need to test it on a broad range of possible inputs
  - Our code needs to work on **all** possible test cases, not just a specific set!

# How can we use testing to find bugs or confirm our code runs properly? What are some strategies to write comprehensive test cases?

- To be confident our code is correct, we need to test it on a broad range of possible inputs
  - Our code needs to work on **all** possible test cases, not just a specific set!
- Think about the different parts of your code and ensure you have a test case to test each of your code's components
  - We can't usually test **all possible inputs** - instead we test each **category** of input

# How can we use testing to find bugs or confirm our code runs properly? What are some strategies to write comprehensive test cases?

- To be confident our code is correct, we need to test it on a broad range of possible inputs
  - Our code needs to work on **all** possible test cases, not just a specific set!
- Think about the different parts of your code and ensure you have a test case to test each of your code's components
  - We can't usually test **all possible inputs** - instead we test each **category** of input
  - Don't forget to test corner cases!

# Find the errors! Classify them as syntax, run-time or logic and fix them with a correct line of code

(a)
```
def disemvowel(text):
    """ Returns string `text` with all vowels removed """
    vowels = ('a', 'e', 'i', 'o', 'u')
    answer = text[0]
    for char in text:
        if char.lower() is not in vowels:
            answer = char + answer
    print(answer)
```

# Find the errors! Classify them as syntax, run-time or logic and fix them with a correct line of code

(a)
```
1  def disemvowel(text):
2      """ Returns string `text` with all vowels removed """
3      vowels = ('a', 'e', 'i', 'o', 'u')
4      answer = text[0]
5      for char in text:
6          if char.lower() is not in vowels:
7              answer = char + answer
8      print(answer)
```

- *line 4; logic/run-time (if empty string);* `answer = ''`
- *line 6; syntax;* `if char.lower() not in vowels:`
- *line 7; logic;* `answer = answer + char` *or* `answer += char`
- *line 8; logic;* `return answer`

# Find the errors! Classify them as syntax, run-time or logic and fix them with a correct line of code

(b)
```
def big-ratio(nums, n):
    """ Calculates and returns the ratio of numbers
    in list `nums` which are larger than `n` """
    n = 0
    greater_n = 0
    for number in nums:
        if number > n:
            greater_n += 1
            total += 1
    return greater_n / total

nums = [4, 5, 6]
    low = 4
print(f"{100*big_ratio(nums, low)}% of numbers are greater than {low}")
```

# Find the errors! Classify them as syntax, run-time or logic and fix them with a correct line of code

(b)
```
1  def big-ratio(nums, n):
2      """ Calculates and returns the ratio of numbers
3      in list `nums` which are larger than `n` """
4      n = 0
5      greater_n = 0
6      for number in nums:
7          if number > n:
8              greater_n += 1
9              total += 1
10     return greater_n / total
11
12 nums = [4, 5, 6]
13     low = 4
14 print(f"{100*big_ratio(nums, low)}% of numbers are greater than {low}")
```

- *line 1; syntax;* `def big_ratio(nums,n):`
- *line 4; logic/(run-time as well since it would cause error as* `total` *is undefinedl);* `total = 0`
- *line 9; logic; remove one level of indentation (outside* `if` *block)*
- *line 13; syntax; remove indentation*

# This function should take a list of integers and remove the negative integers from the list, but it doesn't work :(

```
def remove_negative(nums):
    for num in nums:
        if num < 0:
            nums.remove(num)
```

- Write down three test cases that will help us find the bug
- Debug the code snippet to solve the problem

**Previous exam question:** this code validates a data entry, a list with the following string elements

(a) a *staff ID*, valid if it is a 5 digit number (e.g. `"00520"` or `"19471"`)

(b) a *first name*, valid if non-empty and only containing alphabetical letters

(c) a *password*, valid if including at least one lower-case letter, one upper-case letter, and one punctuation mark from the following `[',', '.', '!', '?']`

The function should return `True` if the data entry contains entirely valid values (according to the above rules) and `False` if any of the fields are invalid. A valid data example is: `['10001', 'Chris', 'Comp!']`

**Previous exam question:** this code validates a data entry, a list with the following string elements

(a) a *staff ID*, valid if it is a 5 digit number (e.g. "00520"

(b) a *first name*, valid if non-empty and only containing a

(c) a *password*, valid if including at least one lower-case l
tion mark from the following [',', '.', '!', '?

The function should return `True` if the data entry contains
rules) and `False` if any of the fields are invalid. A valid data

```
STAFFID_LEN = 5

def validate(data):
    staffid = data.pop(0)
    if not 10**(STAFFID_LEN-1) <= int(staffid) < 10**STAFFID_LEN:
        return False

    first_name = data.pop(0)
    if not first_name and first_name.isalpha():
        return False

    password = data.pop(0)
    contains_lower = contains_upper = contains_punct = False
    for letter in password:
        if letter.islower():
            contains_lower = True
        elif letter.isupper():
            contains_upper = True
        elif not letter.strip(',.!?␣'):
            contains_punct = True

    if not contains_lower and contains_upper and contains_punct:
        return False

    return True
```

# **Previous exam question:** this code validates a data entry, a list with the following string elements

(a) a *staff ID*, valid if it is a 5 digit number (e.g. "00520"

(b) a *first name*, valid if non-empty and only containing a

(c) a *password*, valid if including at least one lower-case l
tion mark from the following [',', '.', '!', '?

The function should return `True` if the data entry contains
rules) and `False` if any of the fields are invalid. A valid data

1. Provide an example of valid data that is correctly classified as such by the provided code (i.e. valid data input where the return value is True).
2. Provide an example of invalid data that is correctly classified as such by the provided code (i.e. invalid data input where the return value is False).
3. Provide an example of invalid data that is incorrectly classified as a valid by the provided code (i.e. valid data input where the return value is erroneously True).
4. Provide an example of valid data that is incorrectly classified as an invalid by the provided code (i.e. invalid data input where the return value is erroneously False).

```
STAFFID_LEN = 5

def validate(data):
    staffid = data.pop(0)
    if not 10**(STAFFID_LEN-1) <= int(staffid) < 10**STAFFID_LEN:
        return False

    first_name = data.pop(0)
    if not first_name and first_name.isalpha():
        return False

    password = data.pop(0)
    contains_lower = contains_upper = contains_punct = False
    for letter in password:
        if letter.islower():
            contains_lower = True
        elif letter.isupper():
            contains_upper = True
        elif not letter.strip(',.!?␣'):
            contains_punct = True

    if not contains_lower and contains_upper and contains_punct:
        return False

    return True
```