Week 7 Tutorial

COMP10001 – Foundations of Computing

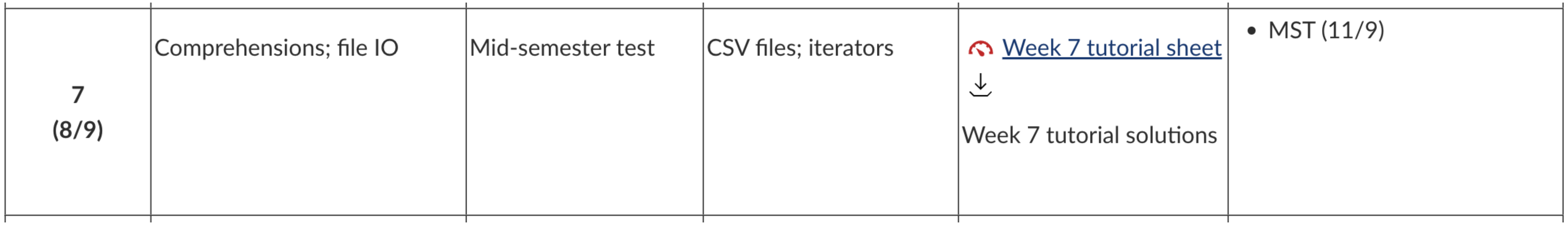
Semester 2, 2025

Clement Chau

- Code structure, Variable names, Test cases
- Docstring and Commenting

Agenda

1. Week 7 Discussion – **Tutorial sheet** (~ 55 mins)
2. One-on-one Q&A (~ 55 mins)

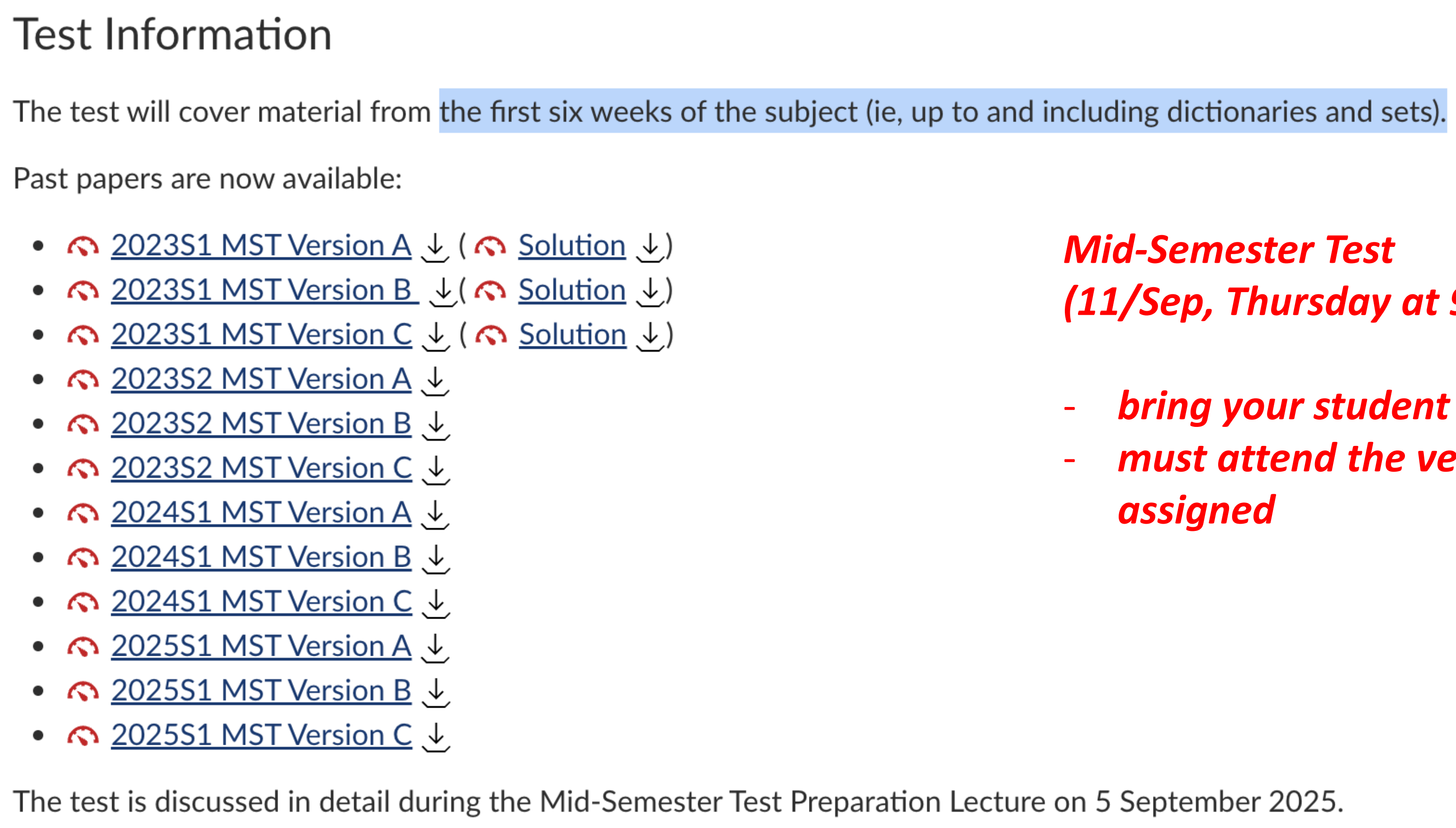


7 (8/9)	Comprehensions; file IO	Mid-semester test	CSV files; iterators	Week 7 tutorial sheet Week 7 tutorial solutions	• MST (11/9)

Mid-Semester Test	***(11/Sep, Thursday at 9 am)***
Ed worksheets 10, 11 due	***(15/Sep, Monday at 6 pm)***

Mid-Semester Test

Canvas > COMP10001_2025_SM2 > Pages > ***MST and Exam [Link]***



Test Information

The test will cover material from the first six weeks of the subject (ie, up to and including dictionaries and sets).

Past papers are now available:

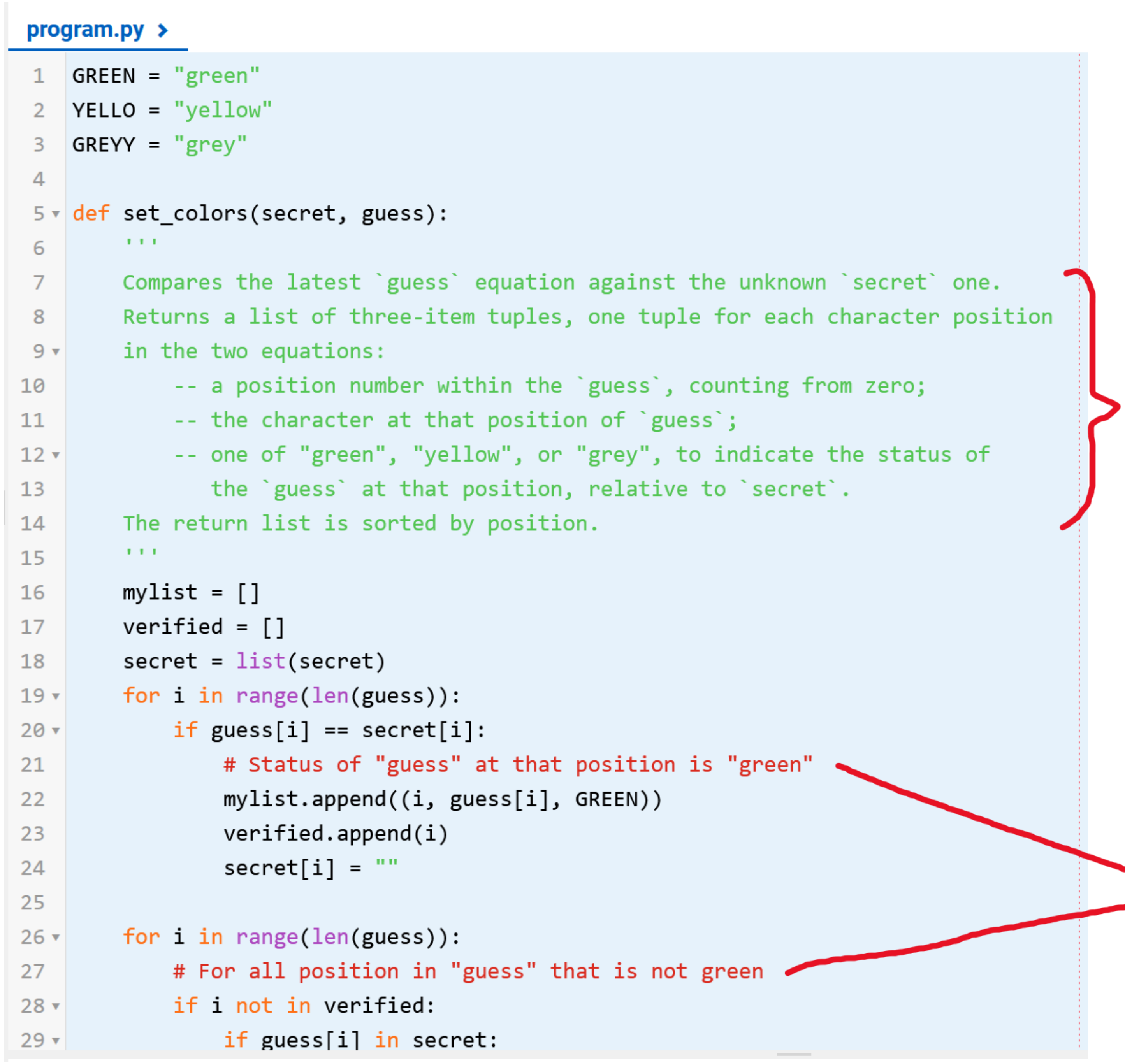
- 2023S1 MST Version A (Solution)
- 2023S1 MST Version B (Solution)
- 2023S1 MST Version C (Solution)
- 2023S2 MST Version A
- 2023S2 MST Version B
- 2023S2 MST Version C
- 2024S1 MST Version A
- 2024S1 MST Version B
- 2024S1 MST Version C
- 2025S1 MST Version A
- 2025S1 MST Version B
- 2025S1 MST Version C

The test is discussed in detail during the Mid-Semester Test Preparation Lecture on 5 September 2025.

Mid-Semester Test
(11/Sep, Thursday at 9 am)

- ***bring your student ID***
- ***must attend the venue/seat to which you have been assigned***

What kind of "commenting" are we expecting in this subject?



program.py

```python
GREEN = "green"
YELLO = "yellow"
GREYY = "grey"

def set_colors(secret, guess):
    '''
    Compares the latest `guess` equation against the unknown `secret` one.
    Returns a list of three-item tuples, one tuple for each character position
    in the two equations:
        -- a position number within the `guess`, counting from zero;
        -- the character at that position of `guess`;
        -- one of "green", "yellow", or "grey", to indicate the status of
           the `guess` at that position, relative to `secret`.
    The return list is sorted by position.
    '''
    mylist = []
    verified = []
    secret = list(secret)
    for i in range(len(guess)):
        if guess[i] == secret[i]:
            # Status of "guess" at that position is "green"
            mylist.append((i, guess[i], GREEN))
            verified.append(i)
            secret[i] = ""

    for i in range(len(guess)):
        # For all position in "guess" that is not green
        if i not in verified:
            if guess[i] in secret:
```

Docstrings are needed to describe what your **function** does at a high-level.

Describes **the input**, **what the function does with the input**, and the **output returned**.

Sufficient **Comments** are needed throughout your program to describe **not-as-obvious parts** of the program.

TuteSheet Week 7 – Past Project

1. For this part of the tutorial you'll be putting yourself into your tutor's shoes and "marking" some solutions. The solutions are to the problem:

   > Write a function `get_species_richness()` that calculates the species richness of a habitat, based on a series of observations of various bird species. The function takes one argument: `observed_list`, a list of independent observations of bird species. The function should return a tuple consisting of:
   >
   > - the species richness, calculated as the number of different species observed
   > - an alphabetically sorted list of the species that were observed

TuteSheet Week 7 – Past Project 1-(1)

(a) Is the author's approach sensible? Did they over-complicate it with unnecessary or convoluted parts? Did they over-simplify it by taking shortcuts that don't work or forgetting requirements?

__Excellent__. Used suitable data structures and __built-in functions__ to simplify their approach. This solution does everything required in the instructions. Not over-simplified anything.

(b) Has the author followed PEP8 guidelines? Did they use descriptive variable names? Are there any "magic numbers" or other unexplained literals?

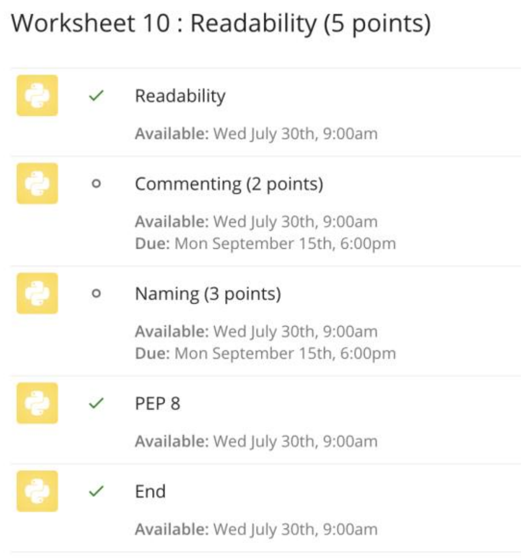
Follows the __PEP8 guidelines__. Sensibly named __variables__. No __magic numbers__ or No __unexplained literals.__

(c) How is the commenting in the solution? Did the author include a docstring?

The commenting is what lets this solution down. A better comment : __why the `observed_list` was being converted to a set__. Missing a __docstring__.

Solution 1:

```python
def get_species_richness(observed_list):
    # easy approach is to convert to a set
    species_observed = set(observed_list)
    return len(species_observed), sorted(species_observed)
```



Worksheet 10 : Readability (5 points)

✓ Readability
Available: Wed July 30th, 9:00am

○ Commenting (2 points)
Available: Wed July 30th, 9:00am
Due: Mon September 15th, 6:00pm

○ Naming (3 points)
Available: Wed July 30th, 9:00am
Due: Mon September 15th, 6:00pm

✓ PEP 8
Available: Wed July 30th, 9:00am

✓ End
Available: Wed July 30th, 9:00am

TuteSheet Week 7 – Past Project 1-(2)

Marking the solution 2

(a) Sensible approach? *__Excellent__, although it is not as short as the first. This solution __does everything required__ in the instructions. __Well-structured.__*

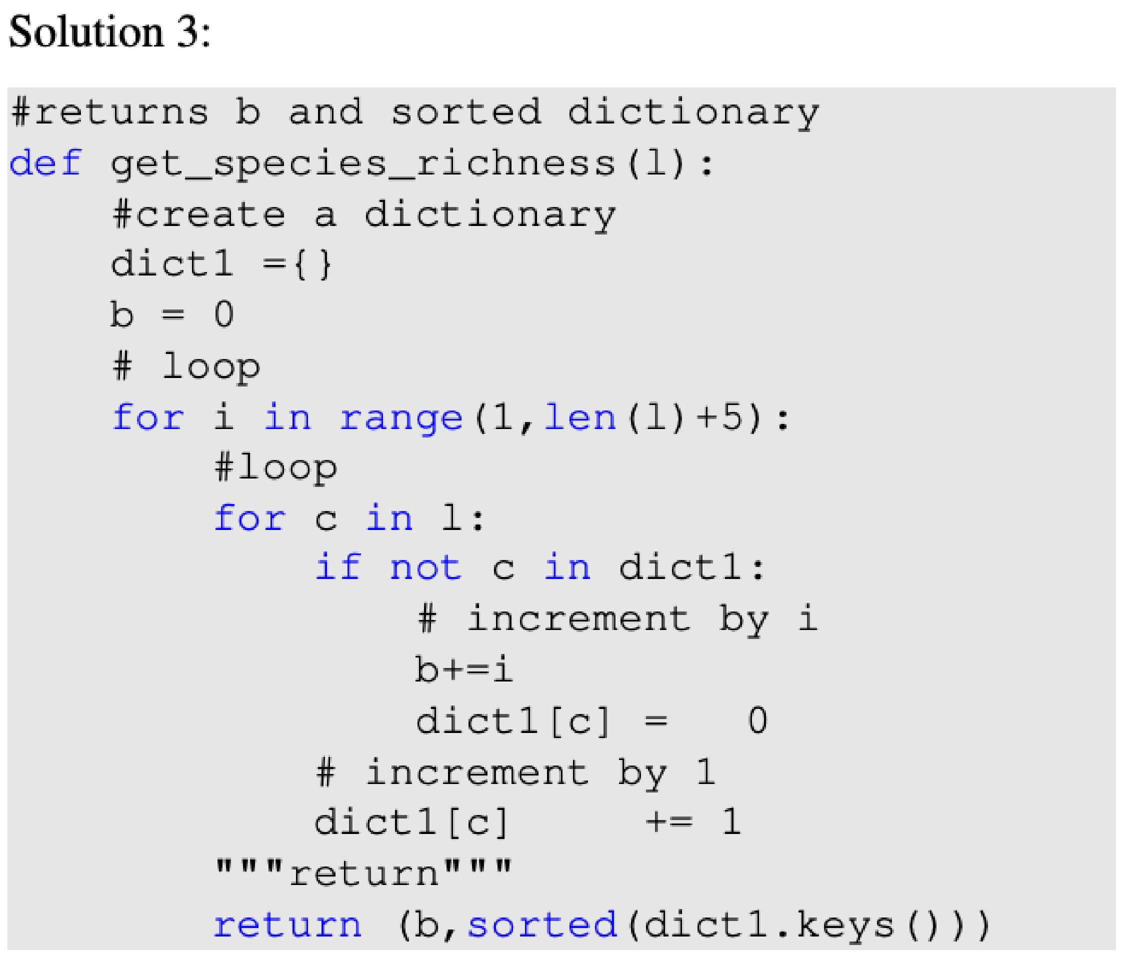
(b) PEP8 guidelines? *Follows the __PEP8 guidelines__. Sensibly named __variables__. No __magic numbers__ or No __unexplained literals__.*

(c) Commenting? *__Excellent__. Useful comments that explains __why each block of code exists__ in their solution. Includes a __docstring__ which tells us the inputs, outputs and brief summary of what the function does.*

Solution 2:

```
def get_species_richness(observed_list):
    """ Takes a list of strings representing species observed. Returns a
    tuple containing the species richness (an int) and a sorted list
    containing names of each species. """
    observed_birds = []

    # constructs unique list of observed bird species
    for bird in observed_list:
        if bird not in observed_birds:
            observed_birds.append(bird)

    # counts number of species and sorts by unicode sort order
    return (len(observed_birds), sorted(observed_birds))
```

TuteSheet Week 7 – Past Project 1-(3)


Solution 3:

```python
#returns b and sorted dictionary
def get_species_richness(l):
    #create a dictionary
    dict1 ={}
    b = 0
    # loop
    for i in range(1,len(l)+5):
        #loop
        for c in l:
            if not c in dict1:
                # increment by i
                b+=i
                dict1[c] =   0
            # increment by 1
            dict1[c]     += 1
        """return"""
        return (b,sorted(dict1.keys()))
```

Marking the solution 3

(a) Sensible approach?

__Over-complicated__ and __difficult to read__.
Correct in terms of test-case correctness, however, need to work on __simplifying__ and more __readable__.

(b) PEP8 guidelines?

Several __PEP8 violations__. __Spacing__ around operators and __commas__ is inconsistent. __variable__ names (not descriptive). Used __magic numbers__

(c) Commenting?

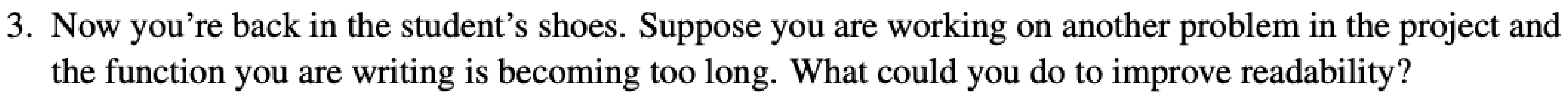
Many of them are __unhelpful__. repeating the code.
No explanation why coding decisions were made.
Not properly included a docstring

TuteSheet Week 7 – Past Project 2

2. Construct three more test cases for the `get_species_richness()` problem in the test harness below at the `# TODO` comments. To write test cases, we should think about different possible inputs our code could receive and cover as many of them as possible. This does not mean writing a test case for every possible input, rather a test case for each category of input, especially testing any "corner cases" which are at the limits of the code's specification. We use test cases for marking correctness but testing also needs to be done by the author of the code!

```
# this only runs when pressing 'run', not when 'testing'
if __name__ == "__main__":
    inputs = [
        ['cockatoo', 'magpie'],
        # TODO: add your test case inputs here  # sorting
    ]   [], # empty                              # multiples
    expected_outputs = [                         # longer
        (2, ['cockatoo', 'magpie']),
        # TODO: add the expected outputs of your test cases here
    ]  (0, [])
    for test_input, expected in zip(inputs, expected_outputs):
        print("expected:", expected)
        print("result:  ", get_species_richness(test_input))
```

TuteSheet Week 7 – Past Project 3


3. Now you're back in the student's shoes. Suppose you are working on another problem in the project and the function you are writing is becoming too long. What could you do to improve readability?

Define helper functions!

*A helper function is a function that **performs some part of the computation of another function**.*

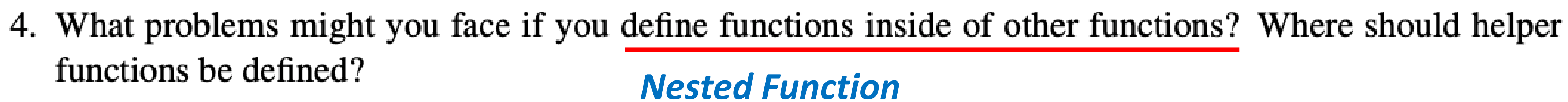
*Helper functions can make programs more **readable** by giving descriptive names to computations.*

By taking computations out of a function and placing them in helper functions, we can then reuse those helper functions if we ever need that computation again.

*This is **much easier than copy-pasting parts of a larger function**.*

*You might also check to see if there are any **redundant parts of your code or if any part is more complex** than it needs to be.*

TuteSheet Week 7 – Past Project 4



4. What problems might you face if you define functions inside of other functions? Where should helper functions be defined?

Nested Function

*Nesting function definitions makes it **harder to reuse them**, as the inner function only exists while the outer function is running.*

*This also means you **won't be able to unit test the inner function**.*

*It can also make your **code confusing** and likely to use variables that you haven't explicitly passed in to your function as an argument (**non-local**). Non-local variables are not great to use because they can make your code harder to understand and debug.*

***Define helper functions outside of other functions**, usually at the top level of your file (alongside other functions). This way they can be used by multiple functions, are easier to test and your code stays clean and readable.*

TuteSheet Week 7 – Question 1

1. Fill in the blanks with comments and a docstring for the following function, which finds the most popular animals by counting ballots. An example for `ballots` is `['dog', 'pig', 'cat', 'pig', 'dog']`, in which case the function returns `['dog', 'pig']`.

```python
def favourite_animal(ballots):
    """ ..."""
    tally = {}

    # ...
    for animal in ballots:
        if animal in tally:
            tally[animal] += 1
        else:
            tally[animal] = 1

    # ...
    most_votes = max(tally.values())
    favourites = []
    for animal, votes in tally.items():
        if votes == most_votes:
            favourites.append(animal)

    return favourites
```

"""Takes a list 'ballots' as input. Counts the frequency of each animal in 'ballots', and returns a list of the most frequently voted animals"""

# Counts frequencies of each animal in the ballots.

# Find and store the animals that received the highest number of votes.

TuteSheet Week 7 – Question 2 (a)

2. Consider the following programs. What are the problematic aspects of their variable names and use of magic numbers? What improvements would you make to improve readability?

➔ *Using constants for the **conversion multipliers** and **appropriate variable names***

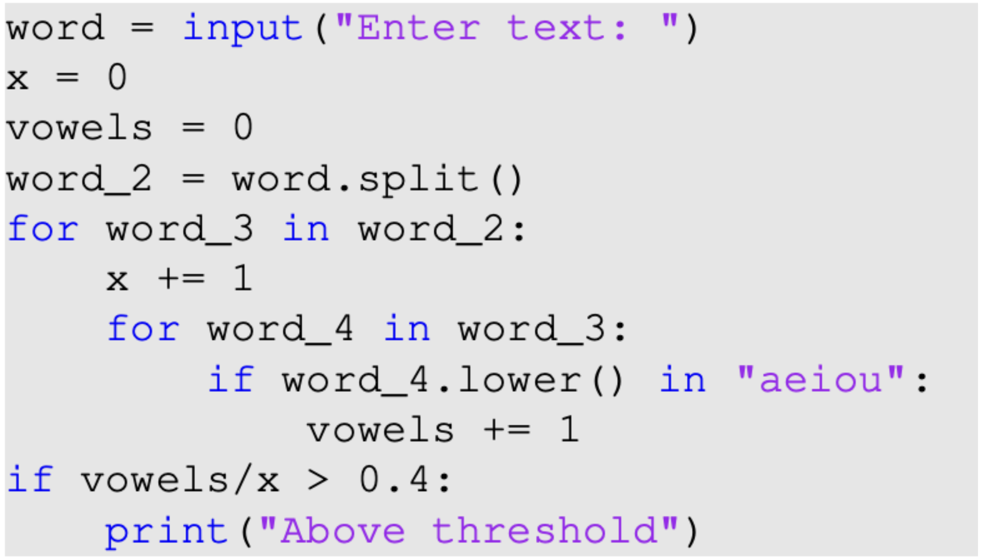
(a)
```
a = float(input("Enter days: "))
b = a * 24   HOUR_DAY = 24
c = b * 60   MINUTE_HOUR = 60
d = c * 60   SECOND_MINUTE = 60
print("There are", b, "hours", c, "minutes", d, "seconds in", a, "days")
```

```
HOUR_DAY = 24
MINUTE_HOUR = 60
SECOND_MINUTE = 60

days = float(input("Enter days: "))
hours = days * HOUR_DAY
minutes = hours * MINUTE_HOUR
seconds = minutes * SECOND_MINUTE

print("There are", hours, "hours", minutes, "minutes", seconds, "seconds in", days, "days")
```

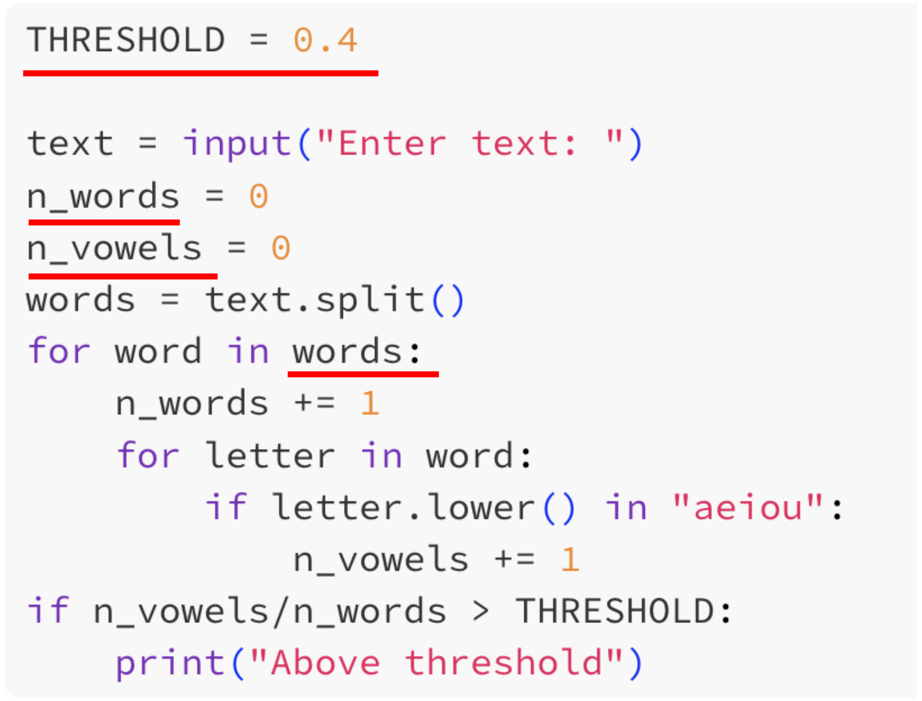
TuteSheet Week 7 – Question 2 (b)

2. Consider the following programs. What are the problematic aspects of their variable names and use of magic numbers? What improvements would you make to improve readability?

➔ *variable names, matching the plurality of nouns*
➔ *The prefix of n_ to variables*
➔ *Instead of a magic number : THRESHOLD*



```python
word = input("Enter text: ")
x = 0
vowels = 0
word_2 = word.split()
for word_3 in word_2:
    x += 1
    for word_4 in word_3:
        if word_4.lower() in "aeiou":
            vowels += 1
if vowels/x > 0.4:
    print("Above threshold")
```



```python
THRESHOLD = 0.4

text = input("Enter text: ")
n_words = 0
n_vowels = 0
words = text.split()
for word in words:
    n_words += 1
    for letter in word:
        if letter.lower() in "aeiou":
            n_vowels += 1
if n_vowels/n_words > THRESHOLD:
    print("Above threshold")
```

TuteSheet Week 7 – Question 3

3. The function below is supposed to take a list of integers and remove the negative integers from the list, however, it is not working as intended.

- Write down three test cases that could be useful for function verification or finding bugs.
- Debug the associated code snippet to solve the problem.

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

# Empty

```
# Test case 1: Empty list
lst = []
remove_negative(lst)
print(lst)
```

```
[]
```

# No/Only negative

```
# Test Case 2: no negative
lst = [0, 1, 2]
remove_negative(lst)
print(lst)
```

```
[0, 1, 2]
```

# Mixed numbers

```
# Test Case 4: mixed numbers
lst = [-1, -2, 3]
remove_negative(lst)
print(lst)
```

```
[-2, 3]
```

Independent Work

- **Next due dates:**
  - Ed Worksheets **10** and **11** is **due Monday, September 15th, 6pm**.
  - Your **Project 1** is **due Friday, September 19th, 6pm**.
- **Raise your hand** if you have any questions!

Scan here for annotated slides