[Introduction 2](#_Toc70521939)

[Chapter 1 3](#_Toc70521940)

[Chapter 2 6](#_Toc70521941)

[Chapter 3 9](#_Toc70521949)

[Chapter 4 14](#_Toc70521950)

[Chapter 5 19](#_Toc70521951)

[Chapter 6 24](#_Toc70521952)

[Chapter 7 27](#_Toc70521953)

[Chapter 8 29](#_Toc70521954)

[Chapter 9 33](#_Toc70521955)

[Chapter 10 35](#_Toc70521956)

[Chapter 11 45](#_Toc70521957)

# Introduction

These case studies are designed to apply what has been learned to real-life scenarios. Although it is broken into chapters so that you can build the case studies bit by bit as you learn, it is likely that you will not have enough time during the course to complete most of the tasks. For this reason, the exercises are written in a way that will enable you to go through them independently. This way you can do them during class to reinforce what you have learned or do them later at a more casual pace to refresh your memory and extend what you have learned.

Some tasks do require knowledge beyond what will be taught in the course. I have tried to keep these to a minimum so we can focus on the basics, but if I have also marked them as ‘challenge’ questions so that you can skip them if you want to.

Alongside this guide, I have also provided the solutions in two formats. The first is a copy of the guide with the solutions (all code and the answers to questions). If you are new to Python, then you might want to work with this version so that you can see the code and try it for yourself. In this way you can focus on the meaning of the code without worrying about the (sometimes fiddly) details. If you are reasonably confident, however, then feel free to work out the code for yourself.

The solutions have also been provided for each chapter in .py files with names based on the question number. This is so that you can run them for yourself to prove that they work, and to see the expected behaviour for yourself. It also means that if you have not completed an exercise you can copy the solutions and move on to the next chapter so that you don’t get left behind.

To cover the topics, we have two case studies. The first is a simple guessing game, where the user is required to guess a number between 1 and 10. This case study focuses on user interaction, strings, loops, and other fundamental concepts.

The second is a more challenging application that allows us to create and manipulate attendees, agenda items, and other details of business meetings. This will focus on functions, collections, classes, exception handling, files, and unit testing.

You will find the final version of each case study in the ‘Final Version’ folder. For the guessing game simply run guess.py, for the calendar application run calnendar\_test.py.

Enjoy your exploration of Python.

# Chapter 1

For this chapter we will focus on some of the code needed for our guessing game.

## Question 1

Import the random module and use help to look at the various functions that are available. We want a function that will allow us to generate a number between 1 and 10. Try out the functions if you are not sure which one to use.

## Question 2

Modify the code so that it saves the random number in a variable.

Get input from the keyboard and compare it with the value in the variable.

Test your code by trying the numbers from 1 to 10 until you guess the number.

Once you have gone through every possible value unsuccessfully print out r and see what the number was. (Yes your code won’t actually work, and hence you will never guess the number, we will fix this in the next question).

## Question 3

Why did the code not work?

1. To illustrate the problem, write code that matches user input with a specific value (e.g. 6) rather than a random number. Type in the value, then check the result.

#### Result:

>>> 6==input()

6

False

>>>

1. So what is going on? Carefully check out the code on page 29 of the manual. What do we need to do with the input before we can use it in a calculation? Fix the above code accordingly to check that it works, then go back and modify our guessing game example.
2. Apply this concept to fix the code for our guessing game:

## Question 4

Improve the code by printing a message. If you correctly guessed the number print “Success!” otherwise print “Try again.” Test the code by running it over and over until you have at least one “Success!” and one “Try again.” Again, test the code until you guess the number.

#### Results (after making repeated guesses and finally guessing 4 – you might have to go as far as 10 before you get the correct guess):

4

Success!

>>>

## Question 5

Now improve the code even more by looping 10 times to give the user ten chances to guess the number.

#### Results (after guessing 1 through to 10):

1

Success!

2

Try again.

3

Try again.

4

Try again.

5

Try again.

6

Try again.

7

Try again.

8

Try again.

9

Try again.

10

Try again.

>>>

## Question 6

Rewrite the code to use a while loop. Once the user has guessed the number, the program should exit the loop. (hint: use a break)

#### Results (after guessing 1,2,3 and 4 – you might have to go as far as 10 before you get the correct guess):

1

Try again.

2

Try again.

3

Try again.

4

Success!

>>>

## Question 7

Modify the code so that the loop will continue until the user successfully guesses the number or gives up by typing 0. (hint: save the input into a variable and use elif to test the various conditions for exiting the loop.) If they give up print out the message ‘Sorry to see you go.’

#### Results (after guessing 1,2,3, and 0):

1

Try again.

2

Try again.

3

Try again.

0

Sorry to see you go.

>>>

# Chapter 2

Open up the code from the previous exercise (or the final solutions file)

>>> import random

>>> r=random.randrange(1,10)

>>> while True:

i=int(input()) #prompt for a guess

if i==0:

print('Sorry to see you go.')

break

elif i!=r:

print('Try again.')

else:

print('Success!')

break

We are going to modify code so that it prints out more user-friendly messages. First we need to become familiar with how to work with strings, then once we a familiar with this we can incorporate it into our application.

## Question 1

Prompt the user for their first and last names and save them in a variable. Then print a message saying .“hello,” along with their full name, making sure it is in capitals

#### Result after typing in ‘matthew’ and then ‘gregory’ should be:

Hello, Matthew Gregory

## Question 2

Print out the following welcome message:

**Welcome to the guessing game.**

**I am thinking of a number between 1 and 10. Can you guess what it is?**

**Instructions:**

**Enter your guess.**

**If you get it wrong, you can try again.**

**If you give up just type 0 to exit.**

## Question 3

Modify the welcome message to include a ▸(unicode 25B8) in front of each line in the instructions.

#### The result should look like this:

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

## Question 3

Modify the main code in our application so that after printing ‘Sorry to see you go’ it also prints a message ‘The number I was thinking of was ‘ along with the value of the random number.

#### The results should look like this (when you have typed in a 0 to give up):

Sorry to see you go.

The number I was thinking of was 4

>>>

0

## Question 4

During the course we will be creating two applications. The simple guessing game app is the one we have been working towards, but we will also develop a more complex applications for scheduling meetings. A meeting will require a list of agenda items and attendees, so with this in mind let’s explore lists.

* 1. Create a new list of agenda items containing “Welcome”, “Review Minutes of last meeting”, “Financial reports”, “Building proposal”, and “Review of OHS standards”
  2. Confirm the contents of our agenda

#### Results:

['Welcome', 'Review Minutes of last meeting', 'Financial reports', 'Building proposal', 'Review of OHS standards']

* 1. We forgot to put “Other Business” on the agenda. Add it to our list and show the contents again.

#### Results:

['Welcome', 'Review Minutes of last meeting', 'Financial reports', 'Building proposal', 'Review of OHS standards', 'Other Business']

* 1. Use a for loop to print out each agenda item on a new line.

#### Results:

# Welcome

# Review Minutes of last meeting

# Financial reports

# Building proposal

# Review of OHS standards

# Other Business

# Question 5

We will use a dictionary to store our meetings by providing a name for the meeting and a list of agenda items.

1. Create a dictionary of meetings consisting of ‘Executive Meeting’ that has the agenda we have already created and a ‘Social Get Together’ which has an empty list of agenda items.
2. Print out our meetings so that each meeting is on a different line.

#### Results:

#### Executive Meeting ['Welcome', 'Review Minutes of last meeting', 'Financial reports', 'Building proposal', 'Review of OHS standards', 'Other Business']

#### Social Get Together []

# Chapter 3

## Question 1

Combine all the necessary code for the guessing game from the previous chapters into a file called guess.py. Include messages asking for first and last names as well as the welcome messages, any import statements, and setting up the random number.

Include comments regularly throughout your code.

We will need to test that guess 0 and a correct guess both work.

1. To test a guess of 0:
2. Run the file using the run menu in IDLE.
3. Type in 0 to make sure the exit works.

#### Result:

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

Your guess (0 to exit):

0

Sorry to see you go.

The number I was thinking of was 7

1. To test a correct guess:
2. You will notice that after running the file we are still in the REPL (shell) window, we can make use of this to run the file from here instead of going back to the file.
3. Import the script, this will also run it.
4. Keep going until you guess the result correctly.
5. Once you have finished close the window to return to the file. Leave the file open for the next question.

#### Result:

>>> import guess

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

Your guess (0 to exit):

1

Success!

## Question 2

1. Define two functions, one called welcome that prints out all the welcome information, and another called guess\_number that does all the work of guessing the number. Test it by using the run menu and calling the two functions

#### Results:

>>> welcome()

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

>>> guess\_number()

Your guess (0 to exit):

1

Try again.

Your guess (0 to exit):

2

Success!

>>>

1. Close the file and the IMPL to return to the OS command line. Navigate to the appropriate Directly where you are developing your files (e.g. cd ‘C: Users\mtthw\Desktop\Python Basics\MyFiles’). Run the script directly from the command prompt using python, noticing that nothing happens.

#### Results:

C: Users\mtthw\Desktop\Python Basics\MyFiles >python3 guess.py

C: Users\mtthw\Desktop\Python Basics\MyFiles >

1. Add code to the script so that the welcome and guess\_number functions runs when you call the from the command line. Make sure that you code does not run when you do an import in the IMPL.

#### Result when called from command prompt:

C: Users\mtthw\Desktop\Python Basics\MyFiles >python3 guess.py

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

Your guess (0 to exit):

0

Sorry to see you go.

The number I was thinking of was 6

C: Users\mtthw\Desktop\Python Basics\MyFiles >

#### Result when called from IMPL:

>>> from guess import \*

>>>

1. Now run the applicaiton in REPL by running both the welcome and guess\_number functions individually.

#### Results:

>>> welcome()

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

>>> guess\_number()

Your guess (0 to exit):

0

Sorry to see you go.

The number I was thinking of was 3

>>>

1. Rewrite this module by putting the code that call the two functions into another function called main(). In the REPL we can use this function to run the whole application, as well as calling this function if we are running it as a script

#### Result when called from command prompt should be the same as before (just make sure)

#### Result when called from IMPL:

>>> from guess import \*

>>> main()

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

Your guess (0 to exit):

0

Sorry to see you go.

The number I was thinking of was 2

>>>

## Question 3

Modify the code so that the instructions for our guessing\_game are only shown when someone calls help().

#### Results:

>>> from guess import \*

>>> help(guess\_number)

Welcome to the guessing game.

I am thinking of a number between 1 and 10. Can you guess what it is?

Instructions:

▸ Enter your guess.

▸ If you get it wrong you can try again.

▸ If you give up just type 0 to exit.

# Chapter 4

## Question 1

1. Modify the welcome function so that it populates a variable called name that concatenates the first and last names together with a space. Use this name in your welcome message.

#### Results:

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go.

The number I was thinking of was 3

>>>

1. Personalise the goodbye message (‘Sorry to see you go’) by using this variable.

#### Result:

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Traceback (most recent call last):

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 4/guess.py", line 52, in <module>

main()

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 4/guess.py", line 49, in main

guess\_number()

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 4/guess.py", line 37, in guess\_number

print('Sorry to see you go '+name)

NameError: name 'name' is not defined

>>>

1. Why didn’t this work? The variable ‘name’ only exists in the welcome function, so we need to give it a global scope. Define the variable ‘name’ at the beginning of the module and give it a value of ‘’.

#### Result:

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go

The number I was thinking of was 3

>>>

1. The code is correct, but it still isn’t working. This is because when we populate the ‘name’ variable it is still creating and populating a local variable. Instruct the welcome function to use the global variable ‘name’. (If you are not sure about what is happening and how to fix it check page 102-103 of the manual).

#### Result:

Please enter your first name:

matthew

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go Matthew Gregory

The number I was thinking of was 4

>>>

## Question 2

1. Modify welcome function to include a middle name. Make sure that your code prompts for a middle name and includes it in the global ‘name’ variable.

#### Results:

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew John Gregory

Your guess (0 to exit):

0

Sorry to see you go Matthew John Gregory

The number I was thinking of was 5

>>>

1. Modify the welcome function so that it only uses the first and last names when it prints out the hello message.

#### Results:

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go Matthew John Gregory

The number I was thinking of was 5

1. Modify the welcome function to accept three parameters for first, middle, and last names. If a name is not given assign them a default of ‘’ (this way our original code will still work just like it did before even though we haven’t passed any parameters). Make sure you code checks that all three names are empty before prompting. Run the code to check that it still works

#### Results:

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go Matthew John Gregory

The number I was thinking of was 8

1. Now check that when we call the welcome function explicitly, we can call it without names and it will prompt for a name or pass across the names, and it will use them.

#### Results:

>>> welcome()

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

>>> welcome('matthew','john','gregory')

Hello, Matthew Gregory

1. Modify the welcome message to print out just the first and last names. What happens if we call the function with two names?

#### Results:

>>>welcome('matthew','gregory')

Hello, Matthew

1. Call the welcome function so that the two values we provide are for the first and last names.

#### Results:

>>> welcome('matthew', lastName='gregory')

Hello, Matthew Gregory

## Question 3

1. Modify the welcome code so that it uses a list (e.g. [‘matthew’,’john’,’gregory’]) as the parameter. The default should now be and empty list []. When prompting for first, middle and last names capitalise the input before appending it to the list. Populate the global variable by looping through the list adding each entry to the name. When printing out the welcome message use the first (0) and last (len-1) names.

#### Results:

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go Matthew John Gregory

The number I was thinking of was 2

>>>

1. Try running the welcome directly without any parameters. Why is it not prompting for a name? (see pages 97-98 if you can’t work it out).

#### Results:

>>> welcome()

Hello, Matthew Gregory

1. Fix the code so that each time you run it without a parameter it will prompt.

#### Results:

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go Matthew John Gregory

The number I was thinking of was 9

>>> welcome()

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

# Chapter 5

## Question 1

Our task is to improve the code we have already written. In our guessing game we are going to modify the welcome function so that it works with strings instead of list and modifies the global variable directly.

1. Set the global variable to the name passed in.
2. If the name is ‘none’ set it to ‘’ (you can’t concatenate to a NoneType) then prompt the user for the names and use += to concatenate them directly to the global variable (with a space where appropriate).
3. Use the title function (instead of initialise) to capitalise the name variable.
4. In order to print out the first and last names, use split to create a list of names and use slicing to fetch back the first and last values and format (instead of concatenation) to construct the welcome message. (This is better than just concat because it is clear which parts of the string are variable and which parts are constant and is also easier to maintain.).
5. Make sure your code still works. Also test the code by running the welcome function directly passing in a string.

#### Results:

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go Matthew John Gregory

The number I was thinking of was 7

>>> welcome("matthew john gregory")

Hello, Matthew Gregory

>>>

## Question 2

Another alternative to dealing with names in our guessing game might be to use a tuple for the names.

1. In the welcome function, make sure that we do all the work on the names parameter by removing the code that assigns names to name, and changing all the references to names. Just before the final welcome message however, construct the name global variable as a tuple using the first, second and third names from our namesList.

#### Results:

Please enter your first name:

matthew

Please enter your middle name:

john

Please enter your last name:

gregory

Hello, Matthew Gregory

Your guess (0 to exit):

0

Sorry to see you go ('Matthew', 'John', 'Gregory')

The number I was thinking of was 1

>>>

1. Make the ‘Sorry to see you go’ message more user friendly by using tuple unpacking to fetch and use just the first name in the message.

#### Results:

Sorry to see you go Matthew

The number I was thinking of was 5

## Question 3

Check out the calendar\_start.py code, it is based on the work we did with meetings and agenda items in questions 4 and 5 of Chapter 2.

1. Run the code to see what it does.

#### Results:

Current meetings:

Executive Meeting

Agenda Items:

Welcome

Apologies

Review Minutes of last meeting

Financial reports

Building proposal

Review of OHS standards

Other Business

Annual Awards Night

Agenda Items:

Welcome

Dinner Service

Guest Speaker

Award Ceremony

Conclusion

Social Get Together

Agenda Items:

1. In the REPL retrieve back the second agenda item, the third and the fourth items, the first three agenda items, and the last two items from agenda1

#### Results:

'Apologies'

['Review Minutes of last meeting', 'Financial reports']

['Welcome', 'Apologies', 'Review Minutes of last meeting']

['Review of OHS standards', 'Other Business']

1. Modify the printAgenda function so that it includes numbers for the agenda items. Save the file as calendar.py

#### Results:

Current meetings:

Executive Meeting

Agenda Items:

0. Welcome

1. Apologies

2. Review Minutes of last meeting

3. Financial reports

4. Building proposal

5. Review of OHS standards

6. Other Business

Annual Awards Night

Agenda Items:

0. Welcome

1. Dinner Service

2. Guest Speaker

3. Award Ceremony

4. Conclusion

Social Get Together

Agenda Items:

1. To make sure that the index number corresponds with the agenda number, modify the agenda items lists so that the first item is always none. Make sure that this item is not printed out.

#### Results:

Current meetings:

Executive Meeting

Agenda Items:

1. Welcome

2. Apologies

3. Review Minutes of last meeting

4. Financial reports

5. Building proposal

6. Review of OHS standards

7. Other Business

Annual Awards Night

Agenda Items:

1. Welcome

2. Dinner Service

3. Guest Speaker

4. Award Ceremony

5. Conclusion

Social Get Together

Agenda Items:

1. Repeat the slicing we did for part b. with the modified agenda items list.

#### Results:

'Apologies'

['Review Minutes of last meeting', 'Financial reports']

['Welcome', 'Apologies', 'Review Minutes of last meeting']

['Review of OHS standards', 'Other Business']

1. As an alternative to using a loop to print out agenda, try using pretty printing (in the REPL) to print the meetings.

#### Results:

{'Annual Awards Night': [None,

'Welcome',

'Dinner Service',

'Guest Speaker',

'Award Ceremony',

'Conclusion'],

'Executive Meeting': [None,

'Welcome',

'Apologies',

'Review Minutes of last meeting',

'Financial reports',

'Building proposal',

'Review of OHS standards',

'Other Business'],

'Social Get Together': []}

## Question 4

Meetings also need attendees.

1. In calendar.py create a set of 10 attendees for our social event. Feel free to use as any names you want (you can choose class member names, or family and friends, or work collegues, or you can get a random list of names like I did by using <http://listofrandomnames.com/>). Create a copy of these attendees in a list called executive. (We will use these later). In the REPL print the attendees list

#### Results:

>>> print(attendees)

{'Mahalia Marois', 'Vicky Alvarez', None, 'Reatha Riehl', 'Omega Bromley', 'Bella Fullilove', 'Na Brinkman', 'Idell Bumpers', 'Gala Peel', 'Ardella Gaillard', 'Yu Fiedler'}

1. Why is set a better option for attendees than a list?
2. In the REPL, print out attendees in alphabetical order alphabetically.

#### Results:

['Ardella Gaillard', 'Bella Fullilove', 'Gala Peel', 'Idell Bumpers', 'Mahalia Marois', 'Na Brinkman', 'Omega Bromley', 'Reatha Riehl', 'Vicky Alvarez', 'Yu Fiedler']

1. In the REPL, insert “Matthew Gregory” to the list and confirm that he has been added by searching for his name in the set.
2. Check to make sure that the changes you have made to the attendees are NOT reflected in the executive set. Under what circumstances would changes to the attendees set also change the other sets?

#### Results:

>>> executive

{'Idell Bumpers', 'Reatha Riehl', 'Bella Fullilove', 'Yu Fiedler', 'Gala Peel', 'Mahalia Marois', 'Na Brinkman', 'Ardella Gaillard', 'Vicky Alvarez', 'Omega Bromley'}

1. Remove “Matthew Gregory” from the list and confirm that he is no longer there.

## Question 5

Redefine your executive set explicitly so that there is only three people in it (it won’t matter which three, you can choose). Also remove one of the executive from the social attendees list. This will give use a change to practice various set algebra operations.

My example is working with the following sets:

* social = {'Bella Fullilove','Ardella Gaillard','Yu Fiedler','Omega Bromley','Idell Bumpers','Mahalia Marois','Reatha Riehl','Na Brinkman','Vicky Alvarez'}
* executive = {'Gala Peel','Bella Fullilove','Ardella Gaillard'}

1. List all attendees of both meetings. (There should be 10)

#### Results:

{'Omega Bromley', 'Idell Bumpers', 'Na Brinkman', 'Gala Peel', 'Yu Fiedler', 'Bella Fullilove', 'Mahalia Marois', 'Vicky Alvarez', 'Reatha Riehl', 'Ardella Gaillard'}

1. Show me the names of all executives who are not attending the social. (There should be 1)

#### Results:

{'Gala Peel'}

1. Show me all those people who are only attending a single event. (There should be 8)

#### Results:

{'Omega Bromley', 'Idell Bumpers', 'Na Brinkman', 'Gala Peel', 'Yu Fiedler', 'Mahalia Marois', 'Vicky Alvarez', 'Reatha Riehl'}

# Chapter 6

## Question 1

We are going to improve our guessing game by validating the input and print out friendly messages and take appropriate action if the input isn’t valid.

1. Try entering characters as input into the guessing game and note the behaviour.

#### Results:

Your guess (0 to exit):

a

Traceback (most recent call last):

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 6/Question1a.py", line 59, in <module>

main()

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 6/Question1a.py", line 56, in main

guess\_number()

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 6/Question1a.py", line 41, in guess\_number

i=int(input())

ValueError: invalid literal for int() with base 10: 'a'

>>>

1. Fix this issue so that the user receives a more user-friendly message and doesn’t cause the application to crash. (The user should be allowed to make another guess so you need to take care about where you handle the exception).

#### Results:

#### Please enter your first name:

#### matthew

#### Please enter your middle name:

#### john

#### Please enter your last name:

#### gregory

#### Hello, Matthew Gregory

#### Your guess (0 to exit):

#### a

#### Please enter a valid number.

#### Your guess (0 to exit):

#### 0

#### Sorry to see you go Matthew

#### The number I was thinking of was 7

1. Add the details of the error to the message so that we know the details of the error that occurred.

#### Results:

Your guess (0 to exit):

a

The following error occured: invalid literal for int() with base 10: 'a'

Please enter a valid number.

## Question 2

Make sure that if the user doesn’t enter values between 1 and 9 it gives a value error.

1. Run the current application with values of -1 and 10 and notice that it is quite happy to accept these values even though they are not valid.

#### Results:

Your guess (0 to exit):

-1

Try again.

Your guess (0 to exit):

10

Try again.

Your guess (0 to exit):

0

Sorry to see you go Matthew

The number I was thinking of was 7

1. Raise a value error exception with an appropriate message if the input is not between 1 and 9 (hint: you can use the range, see p121, as a shortcut). Don’t forget to allow the value of 0 so that users can exit, but not the value of 10. The current exception handler should be sufficient to deal with this new value error.

#### Results:

Your guess (0 to exit):

-1

The following error occurred: Number not between 1 and 9

Please enter a valid number.

Your guess (0 to exit):

10

The following error occurred: Number not between 1 and 9

Please enter a valid number.

Your guess (0 to exit):

0

Sorry to see you go

The number I was thinking of was 4

1. Instead of an ordinary message, print the error using the standard error output. The resulting message should be in red. (If you are not sure how to do this check out page 180).

#### Results:

Your guess (0 to exit):

a

invalid literal for int() with base 10: 'a'

Please enter a valid number.

Your guess (0 to exit):

11

Number not between 1 and 9

Please enter a valid number.

Your guess (0 to exit):

0

Sorry to see you go

The number I was thinking of was 1The number I was thinking of was 6

# Chapter 7

## Question 1

Write a function that generates a list of all the meetings that contain a particular agenda item. (For brevity, use the comprehension syntax).

1. Start by writing a function that uses comprehension to return a list of all meetings in the dictionary of meetings (ignore the parameter for now).

#### Results:

>>> findMeetings("Welcome")

['Executive Meeting', 'Annual Awards Night', 'Social Get Together']

1. Now add the filtering condition to your comprehension that checks that the agenda of the meeting contains the string that was passed in.

#### Results:

>>> findMeetings("Welcome")

['Executive Meeting', 'Annual Awards Night']

## Question 2

Create a function that displays which lists a given attendee belongs to.

1. Define a generator called getCategory that accepts the name of a person and yields ‘Executive’ if they belong to the executive list, and ‘Attendee’ if they belong to the attendee list.
2. To test you generator first print out your attendees and executive list (so you know who is in which list)

#### Results:

>>> executive

{'Gala Peel', 'Bella Fullilove', 'Ardella Gaillard'}

>>> attendees

{'Idell Bumpers', 'Ardella Gaillard', 'Mahalia Marois', 'Reatha Riehl', 'Vicky Alvarez', 'Yu Fiedler', 'Na Brinkman', 'Omega Bromley', 'Bella Fullilove'}

1. Create an iterator using our generator and a person who is in both list.

#### Results:

>>> c=findCategory('Bella Fullilove')

1. Run next to fetch the next list that this person belongs to, until we have no more lists.

#### Results:

>>> next(c)

'Atteendee'

>>> next(c)

'Executive'

>>> next(c)

Traceback (most recent call last):

File "<pyshell#22>", line 1, in <module>

next(c)

StopIteration

1. Recreate the iterator (you will see why very soon), and this time construct a comprehension to generate a list of categories this person belongs to.

#### Results:

>>> c=findCategory('Bella Fullilove')

>>> [category for category in c]

['Atteendee', 'Executive']

1. Once you have finished using an iterator it can not be reused. Rerun the previous instruction to show this.

Results:

>>> [category for category in c]

[]

1. Hence, instead of saving the iterator to a variable it is usually best to call the generator directly. Rerun the previous command by using the findCategory function instead of the variable c.

#### Results:

>>> [category for category in findCategory('Bella Fullilove')]

['Atteendee', 'Executive']

1. Of course, this applies only to generators themselves. You should save the list itself to a variable if you want to reuse it. Save this list to a variable with an appropriate name

#### Results:

>>> bella =[category for category in findCategory('Bella Fullilove')]

1. Test that our iterator is working by generating appropriately named lists for someone who is only in the executive list, the attendees list, and neither. Check the contents of each list.

#### Results:

>>> gala = [category for category in findCategory('Gala Peel')]

>>> yu = [category for category in findCategory('Yu Fiedler')]

>>> matthew = [category for category in findCategory('Matthew Gregory')]

>>> bella

['Atteendee', 'Executive']

>>> gala

['Executive']

>>> yu

['Atteendee']

>>> matthew

[]

# Chapter 8

## Question 1

Up till now meetings have been a dictionary mapping meeting names and a list of agenda items. Now we want to include attendees in our meetings. I would be difficult to do this with a dictionary so we will instead define our own datatype called meeting that will include a name, a list of agenda items, and a set of attendees, as well as a lot of the functionality we have already defined.

1. At the end of our application (just before we print the meetings), define a new class called meeting. For now, leave the class blank. Test that you can create a meeting and that it’s datatype is ‘Meeting’

#### Results:

>>> m=Meeting()

>>> type(m)

<class '\_\_main\_\_.Meeting'>

1. Create a class invariant to set the name of our meeting when we initialise it, and return the name of the meeting when requested. Test your code.

#### Results:

>>> m=Meeting("Executive Meeting")

>>> m.name()

'Executive Meeting'

1. Repeat these steps for agenda and attendees and test that you can create a meeting called ‘test’ adding agenda1 to the agenda and the executive to the attendees and fetching them back using the appropriate functions.

#### Results:

>>> m=Meeting('test',agenda2,executive)

>>> m.name()

'test'

>>> m.agenda()

[None, 'Welcome', 'Dinner Service', 'Guest Speaker', 'Award Ceremony', 'Conclusion']

>>> m.attendees()

{'Bella Fullilove', 'Ardella Gaillard', 'Gala Peel'}

1. Modify the meetings dictionary so that it is a list of meetings instead of a dictionary of names and agenda items. (You will need to move this code so that it happens AFTER you have defined the meeting class). Why do we get an error when we try to run our code?

#### Results:

Current meetings:

<\_\_main\_\_.Meeting object at 0x00000193731B7250>

Traceback (most recent call last):

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 8/Question1d.py", line 48, in <module>

printMeetings(meetings)

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 8/Question1d.py", line 18, in printMeetings

printAgenda(meetings[meeting])

TypeError: list indices must be integers or slices, not Meeting

1. Modify the printMeeting function to accommodate the changes we have made to our meetings type.

#### Results:

Current meetings:

Executive Meeting

Agenda Items:

1. Welcome

2. Apologies

3. Review Minutes of last meeting

4. Financial reports

5. Building proposal

6. Review of OHS standards

7. Other Business

Annual Awards Night

Agenda Items:

1. Welcome

2. Dinner Service

3. Guest Speaker

4. Award Ceremony

5. Conclusion

Social Get Together

Agenda Items:

1. Further modify the printMeetings function so that it also prints out the attendees (as a set) before listing the agenda items.

#### Results:

Current meetings:

Executive Meeting

Attendees:

{'Ardella Gaillard', 'Gala Peel', 'Bella Fullilove'}

Agenda Items:

1. Welcome

2. Apologies

3. Review Minutes of last meeting

4. Financial reports

5. Building proposal

6. Review of OHS standards

7. Other Business

Annual Awards Night

Attendees:

{'Bella Fullilove', 'Vicky Alvarez', 'Mahalia Marois', 'Reatha Riehl', 'Yu Fiedler', 'Na Brinkman', 'Ardella Gaillard', 'Omega Bromley', 'Idell Bumpers'}

Agenda Items:

1. Welcome

2. Dinner Service

3. Guest Speaker

4. Award Ceremony

5. Conclusion

Social Get Together

Attendees:

{'Bella Fullilove', 'Vicky Alvarez', 'Mahalia Marois', 'Reatha Riehl', 'Yu Fiedler', 'Na Brinkman', 'Ardella Gaillard', 'Omega Bromley', 'Idell Bumpers'}

Agenda Items:

1. The findMeetings function will also not work. We also want to change it so that it returns the names of all meetings attended by a particular person (as this will be far more useful) instead of meeting containing a particular agenda item. Make the necessary changes.
2. Test the findMeetings function by fetching all the meetings for a person who is attending them all and put it into a variable. Use comprehension to list the names of these meetings.

#### Results:

#### >>> mts=findMeetings('Bella Fullilove')

#### >>> [m.name() for m in mts]

#### ['Executive Meeting', 'Annual Awards Night', 'Social Get Together']

## Question 2

We need to add some functionality to our class to make it easier to use. We will create functions to add and remove attendees by name so that we can manage our meetings.

1. Create a function called addAttendees to the meeting class so that we can add the name of a person to the set of attendees for this meeting. Include a message to say that the attendee has been successfully added.
2. Test by copying the first element of our meeting list into a variable m, show the attendees for this meeting, call the method you just created to add a new attendee to this meeting, and then show the attendees again to make sure they have been added.

#### Results:

>>> m=meetings[0]

>>> m.attendees()

{'Gala Peel', 'Ardella Gaillard', 'Bella Fullilove'}

>>> m.addAttendee('Matthew Gregory')

Attendee Matthew Gregory successfully added

>>> m.attendees()

{'Matthew Gregory', 'Gala Peel', 'Ardella Gaillard', 'Bella Fullilove'}

1. In the REPL print the meetings set.

#### Results:

>>> printMeetings(meetings)

Executive Meeting

Attendees:

{'Matthew Gregory', 'Gala Peel', 'Ardella Gaillard', 'Bella Fullilove'}

Agenda Items:

1. Welcome

2. Apologies

3. Review Minutes of last meeting

4. Financial reports

5. Building proposal

6. Review of OHS standards

7. Other Business

…

1. Why has updating m also updated the meeting in our set?
2. We could have done this directly using m.attendees().add('Matthew Gregory'). Why did we choose to create a new function in our class instead?
3. Create a removeAttendees function to remove a person from the meeting. If the person isn’t in the meeting raise a value error with an appropriate message, otherwise remove them from the meeting and print an appropriate message. As before, test by creating a variable, and removing and attendee as well as someone who isn’t attending.

#### Results:

>>> m=meetings[0]

>>> m.attendees()

{'Bella Fullilove', 'Gala Peel', 'Ardella Gaillard'}

>>> m.removeAttendee('Gala Peel')

Attendee Gala Peel successfully removed

>>> m.attendees()

{'Bella Fullilove', 'Ardella Gaillard'}

>>> m.removeAttendee('Matthew Gregory')

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

m.removeAttendee('Matthew Gregory')

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 8/Question2f.py", line 51, in removeAttendee

raise ValueError('Person not found')

ValueError: Person not found

# Chapter 9

## Question 1

We can already find out the details of which meetings a particular person is attending. One thing our application should do is print this out to an appropriately named file.

1. Create a function called writeMeetings. At this point just have it create a file called ‘meetings.txt’. Use utf-8 encoding and overwrite any existing file that has the same name. Don’t forget the code to close the file (even though we haven’t actually written anything to it yet).
2. Run the code using the name of someone who is attending ***only one*** of the meetings. Use notepad (other any word processor) to open the text file that was created. It should be blank. Close the file once you have confirmed this.

#### Results:

>>> writeMeetings('Gala Peel')

#### Contents of the fle:

(The file should be empty)

1. Copy the code from the printMeetings function and paste it in the writeMeetings. Modify this code so that instead of printing it to the screen it writes the contents to the file. There are several things you will need to modify for it to work. (In particular all the print commands will become write commands, new lines will need to be added explicitly when you write a line to the file, meetings will need to be restricted to only those ones that this person attends, and the call to printAgenda will need to be replaced by appropriate code).
2. Rerun the code and make sure that you file contains the details of the meetings.

#### Results:

>>> writeMeetings('Gala Peel')

#### Contents of the fle:

Executive Meeting

Attendees:

{'Ardella Gaillard', 'Gala Peel', 'Bella Fullilove'}

Agenda Items:

1. Welcome

2. Apologies

3. Review Minutes of last meeting

4. Financial reports

5. Building proposal

6. Review of OHS standards

7. Other Business

1. If you want to make sure that print and write meeting functions always do exactly the same thing, it would be best to create a function that returns meetings as a string (formatted with new lines, etc.). You could then share this function to provide the contents for both the printMeetings and writeMeetings functions. (You are welcome to try this, although it is not essential. The code as we have written it in the previous section allows for different content for a file than the screen, which may be the correct solution depending on the requirements of our application)

#### Results:

(Exactly the same as before)

# Chapter 10

We have already been doing a lot of testing each time we create a new function or change the way our application runs. Now we will move all the testing (and associated data) to a different module. This is best practice, if for no other reason than to make sure our testing data and code doesn’t find it’s way accidently into our production code.

## Question 1

Let’s have a look at the testing we did for our meeting class in Question 1 of Chapter 8. We will include those tests as part of our unit testing. Have a quick look at the calendar.py and a new calender\_test\_start.py file in this chapter. All the code to create and show the data has been moved to this module.

Also some of the code in calendar.py has been improved to make the class more useful.

* The printAgenda function is now in the Meeting class.
* We have added a new function printMeeting() that prints an individual meeting.
* The printMeetings() function now just loops through each meeting calling this printMeeting() function for that meeting.
* Since the list of meetings is no longer built in to the calendar module, the find and write meetings have been modified. Now you have to pass in the list if you want to find which meetings a person is attending, or you want to write your meetings to a file.

1. Run the calendar.py module to make sure that nothing is being printed out when we import the code. Now run the calendar\_test\_start.py module to make sure that it prints out all the meetings.

#### Result of calendar.py:

>>>

#### Result of calendar\_test.py:

Executive Meeting

Attendees:

{'Ardella Gaillard', 'Gala Peel', 'Bella Fullilove'}

Agenda Items:

1. Welcome

2. Apologies

3. Review Minutes of last meeting

4. Financial reports

5. Building proposal

6. Review of OHS standards

7. Other Business

Annual Awards Night

Attendees:

{'Yu Fiedler', 'Mahalia Marois', 'Vicky Alvarez', 'Ardella Gaillard', 'Reatha Riehl', 'Omega Bromley', 'Idell Bumpers', 'Na Brinkman', 'Bella Fullilove'}

Agenda Items:

1. Welcome

2. Dinner Service

3. Guest Speaker

4. Award Ceremony

5. Conclusion

Social Get Together

Attendees:

{'Yu Fiedler', 'Mahalia Marois', 'Vicky Alvarez', 'Ardella Gaillard', 'Reatha Riehl', 'Omega Bromley', 'Idell Bumpers', 'Na Brinkman', 'Bella Fullilove'}

Agenda Items:

1. We will now formalise the calendar\_test\_start.py so that it uses unittest to do the testing. Create your own calendar\_test.py module. Import unittest and when the script runs call the unittest.main() function instead of printing out all the meeting details.

#### Results:

----------------------------------------------------------------------

Ran 0 tests in 0.000s

OK

1. We are testing the Meeting class so we need to Import it from the calendar module. We will also create a class to test our module. Call it TestMeetings passing in unittest.TestCase as a parameter. Leave it blank for now. You will notice it runs the module, but it hasn’t performed any test yet.

#### Results:

----------------------------------------------------------------------

Ran 0 tests in 0.000s

OK

1. Define a test case function to confirm that you can call the Meeting constructor. Place the meeting in a variable called m. Use the meeting name of ‘test’ with an empty set for the attendees and an empty list for the agenda. Once you have done this, run it to make sure that we pass the test.

#### Results:

.

----------------------------------------------------------------------

Ran 1 test in 0.003s

OK

>>>

## Question 2

Now we want to test the name() function. We will start by using ==. This is a common mistake, so it is worth thinking about why this won’t work and hence why we need to use assertions.

1. Add code to our test to make sure that we can run the name() function.

#### Results:

.

----------------------------------------------------------------------

Ran 1 test in 0.012s

OK

1. Now try to test that it returns a value of ‘test’. As a first attempt use ==.

#### Results:

.

----------------------------------------------------------------------

Ran 1 test in 0.012s

OK

1. It seems to work but modify your code to see if it returns a value of ‘tst’ instead. Observe that it still passes the test! Why?

#### Results:

.

----------------------------------------------------------------------

Ran 1 test in 0.028s

OK

1. Change your code to use assertEqual. Make sure the test now fails.

#### Results:

F

======================================================================

FAIL: test\_create\_meetings (\_\_main\_\_.TestMeetings)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 10/Question2d.py", line 11, in test\_create\_meetings

self.assertEqual(m.name(),'tst')#use assertions to test behaviour

AssertionError: 'test' != 'tst'

- test

? -

+ tst

----------------------------------------------------------------------

Ran 1 test in 0.039s

FAILED (failures=1)

1. Once you have tested that ‘tst’ fails, change it to ‘test’ and make sure it succeeds.

#### Results:

.

----------------------------------------------------------------------

Ran 1 test in 0.014s

OK

## Question 3 (Challenge Question)

One test we did in our code was to make sure that m is a meeting. This is a challenge question because it is not covered in the course and will need to be investigated.

1. Research documentation to see how you might do this. Add this to our test. Feel free to experiment to make it work.

#### Results:

.

----------------------------------------------------------------------

Ran 1 test in 0.013s

OK

1. To make sure that the test is actually testing the right thing, change the type to str and check that if fails. Once you have confirmed that m is not a string change it back to Meeting.

#### Results:

F

======================================================================

FAIL: test\_create\_meetings (\_\_main\_\_.TestMeetings)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:\Users\mtthw\Desktop\Business\Courses\DWT\Python\Python Basics\Exercise Files\Solutions\Chapter 10\Question3b.py", line 14, in test\_create\_meetings

self.assertIsInstance(m,(str))#Test that it fails

AssertionError: <calendar.Meeting object at 0x000001F04428B2B0> is not an instance of <class 'str'>

----------------------------------------------------------------------

Ran 1 test in 0.040s

FAILED (failures=1)

## Question 4

1. Note that we tested the constructor and the name function in the same test. We should probably test them separately. Make a new test function and move the test for the meeting name to this function. Run the test and notice that it fails. Why?

#### Result:

.E

======================================================================

ERROR: test\_name\_function\_behaviour (\_\_main\_\_.TestMeetings)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 10/Question3a.py", line 22, in test\_name\_function\_behaviour

self.assertEqual(m.name(),'test')#fails because m is not defined

NameError: name 'm' is not defined

----------------------------------------------------------------------

Ran 2 tests in 0.044s

FAILED (errors=1)

1. To test data like this, the best practice is to use a ‘fixture’ to define the data. It can then be used in all test. Create a function called setUp (passing self as a parameter) and create our object m here. and empty list called m as an attribute of self (self being the TestMeetings object). Modify the tests so they refer to this attribute. Now your test should succeed.

#### Results:

...

----------------------------------------------------------------------

Ran 2 tests in 0.044s

OK

## Question 5

Let’s investigate fixtures further to set up data for our testing environment. We will then test all the ‘getter’ function from our Meeting class to make sure they work.

1. Copy all of the data from our calendar module and add it to the setUp() function. Make sure that you modify all the names make them an attribute of ‘self’
2. Test that the name of the first meeting in our list of meetings is ‘Executive Meeting’.

#### Result:

...

----------------------------------------------------------------------

Ran 3 tests in 0.047s

OK

1. Test that the list of agenda items match the expected set of data.

#### Result:

...

----------------------------------------------------------------------

Ran 3 tests in 0.047s

OK

1. Test that the attendees is also as expected..

#### Results:

...

----------------------------------------------------------------------

Ran 3 tests in 0.047s

OK

## Question 6

In the remaining exercises we are going to test the module level functions (printMeetings(), writeMeetings(), findMeetings(), etc). Let’s start by testing that the printMeetings() function works.

1. Add a test to check the printMeetings() function. Explain why this test fails.

#### Results:

....E

======================================================================

ERROR: test\_printMeetings\_exist (\_\_main\_\_.TestMeetings)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:\Users\mtthw\Desktop\Business\Courses\DWT\Python\Python Basics\Exercise Files\Solutions\Chapter 10\Question6.py", line 52, in test\_printMeetings\_exist

printMeetings(self.meetings)

NameError: name 'printMeetings' is not defined

----------------------------------------------------------------------

Ran 5 tests in 0.009s

FAILED (errors=1)

1. Fix the import statement so that we have access to every function in the calendar module and rerun the test.

#### Results:

....Executive Meeting

Attendees:

{'Gala Peel', 'Ardella Gaillard', 'Bella Fullilove'}

Agenda Items:

1. Welcome

2. Apologies

3. Review Minutes of last meeting

4. Financial reports

5. Building proposal

6. Review of OHS standards

7. Other Business

Annual Awards Night

Attendees:

{'Omega Bromley', 'Reatha Riehl', 'Mahalia Marois', 'Yu Fiedler', 'Idell Bumpers', 'Vicky Alvarez', 'Ardella Gaillard', 'Na Brinkman', 'Bella Fullilove'}

Agenda Items:

1. Welcome

2. Dinner Service

3. Guest Speaker

4. Award Ceremony

5. Conclusion

Social Get Together

Attendees:

{'Omega Bromley', 'Reatha Riehl', 'Mahalia Marois', 'Yu Fiedler', 'Idell Bumpers', 'Vicky Alvarez', 'Ardella Gaillard', 'Na Brinkman', 'Bella Fullilove'}

Agenda Items:

.

----------------------------------------------------------------------

Ran 5 tests in 0.176s

OK

## Question 7

Often, we have tests that are useful for debugging purposes. We want to keep them in our unit test but most of the time they should be skipped. It is annoying to have to comment/uncomment this code so Python offers a shortcut.

1. Add the line @unittest.skip('Test no longer required') just before the last test and run it to make sure the meetings are no longer printed out.

#### Results:

....s

----------------------------------------------------------------------

Ran 5 tests in 0.018s

OK (skipped=1)

1. Aside from the fact that we are no longer seeing any meetings, how else can you tell a test has been skipped?
2. How can you tell which test was skipped?

## Question 8

We will now test findMeeting().

1. Create test to make sure the findMeetings function works for a person who is attending all the meetings.
2. Once you have confirmed it is working correctly add the code to skip this test.
3. Now create a test that compares the names of the meetings returned by the function (i.e. [m.name() for m in findMeetings(self.meetings,'Bella Fullilove')] ) with the expected output ['Executive Meeting', 'Annual Awards Night', 'Social Get Together']
4. Sometimes you might want to test that it doesn’t match a certain list. Copy the code and see what happens if you test a match against a result you don’t want (e.g. ['Executive Meeting', 'Annual Awards Night'])

#### Results:

.Fs...s

======================================================================

FAIL: test\_findMeetingBehaviour (\_\_main\_\_.TestMeetings)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 10/Question8d.py", line 64, in test\_findMeetingBehaviour

self.assertEqual([m.name() for m in findMeetings(self.meetings,'Bella Fullilove')],

AssertionError: Lists differ: ['Executive Meeting', 'Annual Awards Night', 'Social Get Together'] != ['Executive Meeting', 'Annual Awards Night']

First list contains 1 additional elements.

First extra element 2:

'Social Get Together'

- ['Executive Meeting', 'Annual Awards Night', 'Social Get Together']

? -----------------------

+ ['Executive Meeting', 'Annual Awards Night']

----------------------------------------------------------------------

Ran 7 tests in 0.068s

FAILED (failures=1, skipped=2)

1. Rewrite the test so that it *succeeds* if it doesn’t match.

## Question 9

Next, we want to test that we can add and remove attendees. We will do this by adding and removing attendees from the first meeting in our list and comparing it against expected results.

1. Create a test to make sure that add attendees will work on the first meeting in our list.
2. In the same test make sure the new set of attendees for this meeting is what we would expect.
3. In the same way create a new test to make sure that we can remove an existing employee and that the result will be what we expect.
4. Copy and paste the function you just created and modify it to test for a person who doesn’t exist in this meeting. This test will fail because it will raise an exception.

#### Results:

Attendee Matthew Gregory successfully added

...s...sAttendee Gala Peel successfully removed

.E

======================================================================

ERROR: test\_removeNonExistingAttendee (\_\_main\_\_.TestMeetings)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 10/Question9d.py", line 80, in test\_removeNonExistingAttendee

self.meetings[0].removeAttendee('Matthew Gregory')

File "C:/Users/mtthw/Desktop/Business/Courses/DWT/Python/Python Basics/Exercise Files/Solutions/Chapter 10\calendar.py", line 53, in removeAttendee

raise ValueError('Person not found')

ValueError: Person not found

----------------------------------------------------------------------

Ran 10 tests in 0.105s

FAILED (errors=1, skipped=2)

1. Modify the code to test for this exception. (You may need to do some research on how to use the assertRaises test)

## Question 10

Our last task is to test the functions that writes meeting to a file.

1. Test that the writeMeetings function works with someone who is attending all the meetings to a file.

#### Results:

Attendee Matthew Gregory successfully added

...s...sAttendee Gala Peel successfully removed

...

----------------------------------------------------------------------

Ran 11 tests in 0.103s

OK (skipped=2)

1. Check that the meetings file was created. You can do this manually by checking the operating system for the meetings.txt file, but you should also add code to your function to test that the file exists. You will need to use os.path to do this, so make sure you import it.

#### Results:

Attendee Matthew Gregory successfully added

...s...sAttendee Gala Peel successfully removed

...

----------------------------------------------------------------------

Ran 11 tests in 0.049s

OK (skipped=2)

1. The file is for testing purposes only, so once we have finished the test we should remove it. Where should we do this to make sure that the file is always removed, even if something goes wrong in our code?
2. Write suitable code that also deals with the OSError we might get if the file doesn’t exist because we exited the test prematurely.
3. Run the test one final time to make sure that it runs successfully and check the OS to make sure the file not there after the test has finished.

#### Result:

Attendee Matthew Gregory successfully added

...s...sAttendee Gala Peel successfully removed

...

----------------------------------------------------------------------

Ran 11 tests in 0.097s

OK (skipped=2)

>>>

# Chapter 11

No Questions for this chapter.