



# ***ISAD1000 Assignment***

HUZAIFA KHAN 20188850

CURTIN UNIVERSITY, SEMESTER ONE 2022

PRACTICAL CLASS: WEDNESDAY 10-12; 314.218

## **Introduction:**

The purpose of this report is to display and convey certain software engineering techniques and uses these techniques to describe and implement code correctly. The code that will be used for this software program is Python. document gives a thorough explanation and description on how software engineering techniques were implemented. Each technique influences the other, and various testing techniques were also used to refine and correct the code. The skills that will be shown are as follows: converting a string into upper or lower case (category 1a), identifying whether numeric values are in a given string (category 1b), identify whether a given string is a valid number or not (category 1c), remove any numeric values in a given string and then convert the string to upper case or lower case (category 1d) and converting a number which represents a time given in hours to minutes and vice versa and time given in minutes to seconds and vice versa (category 2c). This document provides a description of the modules created, and touches on the modularity aspects that were applied in the code, as well as two types of testing techniques that were used (black box and white box testing) to implement the test code. Ethics and Professionalism is another key area that this report discusses and describes using the code designed in the report in a large-scale scenario, and what the lack of ethics and professionalism could lead to.

## **Module Descriptions:**

In this description, sub-parts of the program will be described, and it will convey how the modules will successfully do the job that the scenario requires. However, they will all be under one main program, which will be called productioncode.py. In this umbrella program, the modules that will be described are sub-parts of the program, that are all doing similar tasks and using similar functions and techniques to complete those tasks correctly.

### **Module 1: (Category 1a):**

**Name:** Upper and Lowercase Module

**Method A:** convert\_to\_upper

**Method B:** convert\_to\_lower

**Input:** A string is parsed into the module.

**Output A:** A string that has been converted into all upper-case letters.

**Output B:** A string that has been converted into all lower-case letters.

**Assertion A:** Converts a string to one that has been converted into all upper-case letters.

**Assertion B:** Converts a string to one that has been converted into all lower-case letters.

### **Module 2: (Category 1b):**

**Name:** Numeric Value Module

**Method:** contains\_numeric

**Input:** A string is parsed into the module.

**Output:** A string that checks whether it contains a numeric value or not.

**Assertion A:** Converts a string to indicate that there are numeric values present.

**Assertion B:** Converts a string to indicate that there are no numeric values present.

### **Module 3: (Category 1c):**

**Name:** Valid Number Module

**Method:** valid\_number\_checker

**Input:** A string is parsed into the module.

**Output:** A string that checks whether it contains a valid number or not.

**Assertion A:** Converts string to indicate a valid number.

**Assertion B:** Converts string to indicated invalid number.

### **Module 4: (Category 1d):**

**Name:** Numeric Upper and Lower Module

**Method A:** remove\_number\_upper

**Method B:** remove\_number\_lower

**Input:** A string is parsed into a module.

**Output A:** A string that removes a number and has been converted into all upper-case letters.

**Output B:** A string that removes a number and has been converted into all lower-case letters.

**Assertion A:** Converts a string to one that been converted into all upper-case letters with no numbers.

**Assertion B:** Converts a string to one that been converted into all lower-case letters with no numbers.

### **Module 5: (Category 2c):**

*Converting a number which represents a time given in hours to minutes and vice versa, and time given in minutes to seconds and vice versa.*

**Name:** Time changing module

**Method A:** hours\_to\_minutes and minutes\_to\_hours

**Method B:** minutes\_to\_seconds and seconds\_to\_minutes

**Import:** A string is parsed into the module

**Output A:** A string that has been converted from hours to minutes and minutes to hours.

**Output B:** A string that has been converted from minutes to seconds and seconds to minutes.

**Assertion A:** Converts a string to one that has been converted from hours to minutes and vice versa

**Assertion B:** Converts a string to one that has been converted from minutes to seconds and vice versa.

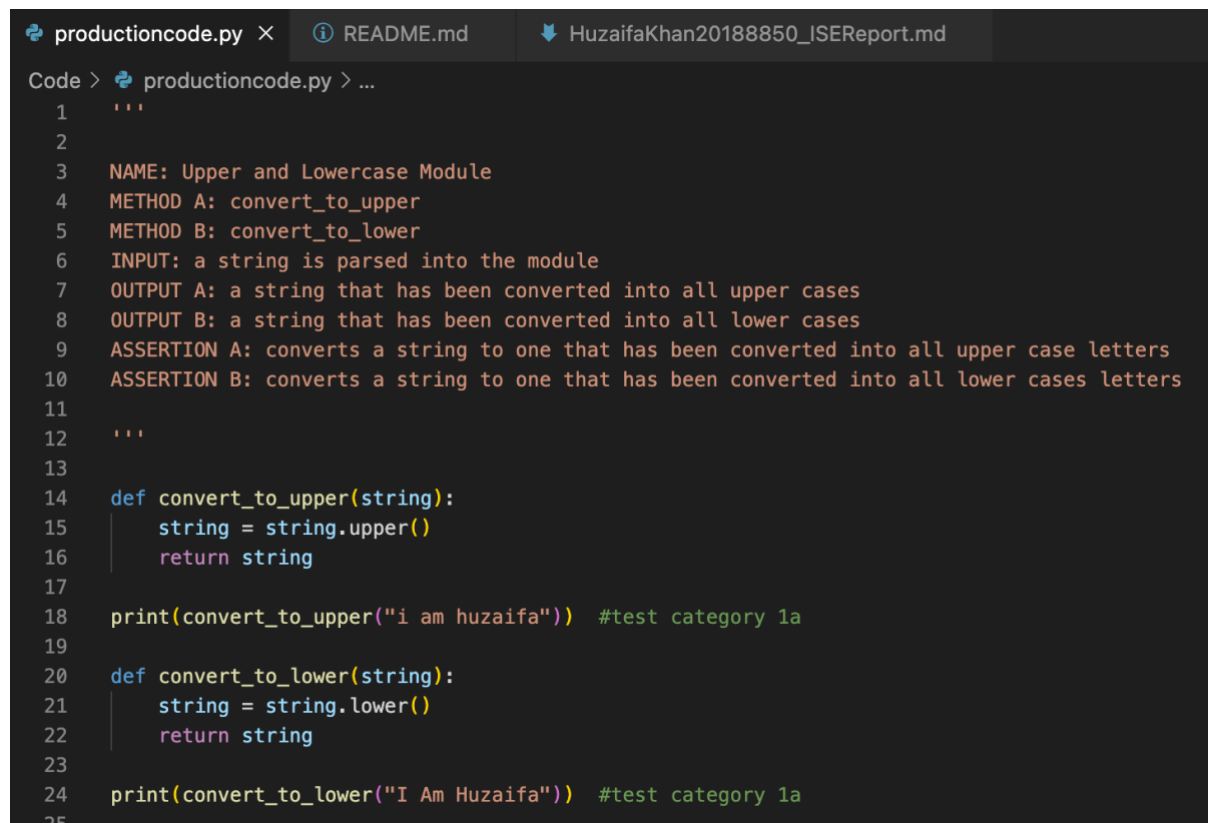
### **Modularity:**

Using the above module descriptions of the modules that have been designed, a program was created using the coding language Python so that each module is performing its task correctly without any errors.

**Description on how to run code with correct commands:**

The code that was implemented based off the module descriptions used the coding language python. For each module, the descriptions on how to run the code is given below.

## Module 1:



```
productioncode.py ×  README.md  HuzaifaKhan20188850_ISEReport.md
Code > productioncode.py > ...
1  '''
2
3  NAME: Upper and Lowercase Module
4  METHOD A: convert_to_upper
5  METHOD B: convert_to_lower
6  INPUT: a string is parsed into the module
7  OUTPUT A: a string that has been converted into all upper cases
8  OUTPUT B: a string that has been converted into all lower cases
9  ASSERTION A: converts a string to one that has been converted into all upper case letters
10 ASSERTION B: converts a string to one that has been converted into all lower cases letters
11
12 '''
13
14 def convert_to_upper(string):
15     string = string.upper()
16     return string
17
18 print(convert_to_upper("i am huzaifa")) #test category 1a
19
20 def convert_to_lower(string):
21     string = string.lower()
22     return string
23
24 print(convert_to_lower("I Am Huzaifa")) #test category 1a
25
```

*Figure 2.1 – Code for Module 1*

The first line of the code “def convert\_to\_upper(string)” is used to define the string that is going to be used in the module. It defines what the method will be in the module, which to “convert\_to\_upper” and defines that for the string. The second line of the code allows the module to convert the string into all upper-case letters, by the function “upper()”, which is part of the intended task of the module. “The return string function” then parses the string through the module. After that, the print() function is used to convey that the task intended has been correctly implemented without any errors when the code is run, and the sentence in the string “i am huzaifa” returns as “I AM HUZAIFA” when the code is run. To convert a string to all lower-case letters, the exact same thing is implemented, however the method is changed from “convert\_to\_upper” to “convert\_to\_lower”, and the function “lower()” is used and parsed through the string.

## Module 2:

In the first line of the code (screenshot of the code is shown below), the string is defined as “def contains\_numeric(string)”. The “number\_in\_string” function allows the string to convey that if it does not contain a numeric number, then it will be “false”. To find whether there are numeric values in the string, we use the isdigit() method. This method returns a true value if there are valid numeric values in the string and returns false if there are no numeric values. Numeric values in this case refer to number only. For example, 20 would be considered a numeric value in the string, however twenty would not. The == signs compare the two sentences that will be checked for numeric values and compares them to see if they give the same value. The “if” indicates whether the expression is true or false; in this case whether

there are numeric values or not. If there are numeric values in the string, the print() functions allow for this to be shown and prints “There are numeric values in the string”. If there aren’t any numeric values, the else function is utilised to figure out what to do when the condition is not met; in this case “there are no numeric values” is printed. The last two lines of the module uses the contains\_numeric to check whether there are, or there are no numeric values in the given sentence.

```
26 '''
27
28 NAME: Numeric Value Module
29 METHOD: contains_numeric
30 INPUT: a string is parsed into the module
31 OUTPUT: a string that checks whether it contains a numeric or not
32 ASSERTION A: converts a string to indicate that there are numeric values present
33 ASSERTION B: converts a string to indicate that there are no numeric values present
34
35 '''
36
37 def contains_numeric(string):
38     number_in_string = False
39     for letter in string:
40         if letter.isdigit() == True:
41             number_in_string = True
42     if number_in_string == True:
43         print("There are numeric values in this string")
44     else:
45         print("There are no numeric values in this string")
46
47 contains_numeric("I am 20 years old") #test category 1b
48 contains_numeric("I am twenty years old") #test category 1b
49
```

*Figure 2.2 – Code for Module 2*

### **Module 3:**

The function “def valid\_number\_checker(string)” is used to define what the function is, and the method “valid\_number\_checker” is parsed through the string. In the second line of the code, isnumeric() is used to identify whether there is a valid numeric value in the given string, and an if statement is used to evaluate whether it is true or false. If there is a valid number, then the print function is used to print “This is a valid number”, otherwise if the criteria is not met, “This is not a valid number” is printed when the code is run. The last two lines of the code use the method “valid\_number\_checker” to check whether the given “numbers” are valid numbers or not; this is indicated when the code is run. The code is shown below.

```

50  '''
51
52  NAME: Valid Number Module
53  METHOD: valid_number_checker
54  INPUT: a string is parsed into the module
55  OUTPUT: a string that checks whether it contains a valid number or an invalid number
56  ASSERTION A: converts a string to indicate a valid number
57  ASSERTION B: converts a string to indicate an invalid number
58
59  '''
60
61  def valid_number_checker(string):
62      if string.isnumeric():
63          print("This is a valid number")
64      else:
65          print("This is not a valid number")
66
67  valid_number_checker("2p67hello")# test category 1 c)
68  valid_number_checker("2670") # test category 1 c)
69
70  '''

```

*Figure 2.3 – Code for Module 3*

#### **Module 4:**

```

'''
NAME: Numeric Upper and Lower Module
METHOD A: remove_number_upper
METHOD B: remove_number_lower
INPUT: a string parsed into the module
OUTPUT A: a string that removes a number and has been converted into all upper cases
OUTPUT B: a string that removes a number and has been converted into all lower cases
ASSERTION A: converts a string to one that has been converted into all upper case letters with no numbers
ASSERTION B: converts a string to one that has been converted into all lower case letters with no numbers

'''

def remove_number_upper(string):
    string = ''.join([i for i in string if not i.isdigit()])
    return string.upper()

def remove_number_lower(string):
    string = ''.join([i for i in string if not i.isdigit()])
    return string.lower()

print(remove_number_upper("I am a 20 twenty years old"))# test category 1 d)

print(remove_number_lower("I am a 20 twenty years old"))# test category 1 d)

```

*Figure 2.4 – Code for Module 4*

The first line of the module defines what method the module will be using and passing the string through the module; in this instance, the method “remove\_number\_upper” is used, which basically removes any numbers from the string and makes all the letters upper case. The second line of the code removes a character (in this case it removes all numeric values) and then joins the remaining string to make one string. This is then returned as one string

with all upper-case letters using the upper() function. The second part of the code does the exact same thing as the first, however it turns the string into all lower-case letters with no numbers. This is then printed for both upper and lower in the last two lines of the code. Numeric values in this module are numeric digits only and do not apply to number that are written, such as “twenty”.

### **Module 5:**

This code offers a lot of reusable code, with only minor differences with each line of code. For the first part, the string is defined, and a simple calculation is written to convert hours to minutes. The return function is used to return hours that have been converted and the print function is used to indicated that there are “ 60 hours in 3600 minutes” when the code is run. The same method is applied when converting minutes to hours, however besides multiplying by 60, we divide by 60 and the print function is used to indicate the given result when the code is run. The same exact functions are used to convert minutes to seconds and vice versa in the second part of the code. A screenshot of module 5 is shown below for a better understanding on how the code is illustrated.

```
'''
NAME: Time Changing Module
METHOD A: hours_to_minutes and minutes_to_hours
METHOD B: minutes_to_seconds and seconds_to_minutes
INPUT: a string parsed into the module
OUTPUT A: a string that has been converted from hours to minutes and minutes to hours
OUTPUT B: a string that has been converted from minutes to seconds and seconds to minutes
ASSERTION A: converts a string to one that has been converted from hours to minutes and vice versa
ASSERTION B: converts a string to one that has been converted from minutes to seconds and vice versa
'''

def hours_to_minutes(hours):
    minutes = hours*60
    return minutes
print("60 hours is " + str(hours_to_minutes(60)) + " minutes")# test category 2 c)

def minutes_to_hours(minutes):
    hours = minutes/60
    return hours
print("60 minutes is " + str(minutes_to_hours(60)) + " hours")# test category 2 c)

def minutes_to_seconds(minutes):
    seconds = minutes*60
    return seconds
print("60 minutes is " + str(minutes_to_seconds(60)) + " seconds")# test category 2 c)

def seconds_to_minutes(seconds):
    minutes= seconds/60
    return minutes
print("60 seconds is " + str(seconds_to_minutes(60)) + " minutes")# test category 2 c)
```

*Figure 2.5 – Code for Module 5*

### Discussion on different modularity concepts applied in code:

While creating the program and writing the code, many modularity techniques were used to improve and make the code more efficient and to the standard of software engineering principles. A divide and conquer approach was used for all the modules. As illustrated in the code, the program runs all at once, however different parts of the code do different tasks; each sub-part of the program performs its own well-defined task. This is clearly shown through using labels above each module, as well as spacing each module out and diving problems into smaller ones and tackling them from the ground up. Every module in the code applies this strategy which made writing the code much easier and allowed for the task of each module to be carried out without many problems. Coupling is a technique that is prevalent in coding. We can see that each module performs its task on its own and is not dependant and reliant on any other module. The content of each module is vaguely connected to each other, for instance similar lines of code might have been used to perform different tasks. However, the internal workings of each module are not shown to another module, and they do not have a significant effect on each other. The code is simple and does not dive into the complexity of python. This shows that there are not that many global variables present, as the use of global variables increase the complexity of the code. This is also shown as in almost every module, return values and parameters are used to import and export information, such as the lines of code shown below.

```
def convert_to_upper(string):  
    string = string.upper()  
    return string  
  
print(convert_to_upper("i am huzaifa")) #test category 1a
```

*Figure 3.1: Snippet from Module*

As shown above, the use of parameters and a return value removes the possibility of having global values. The modules are all very cohesive, as they all perform their specific task precisely and efficiently. As stated above, low/loose coupling is present in the program, which goes hand in hand with high cohesion. Each task is separated into its separate modules that are labelled and illustrated, so that code does not get mixed up and the task of each module is clearly illustrated as well; this indicates high cohesion. The tasks of each module are related to each other in the sense that they all use the same sort of data, for instance they all parse a string through the module and use return values, parameters and if and else statements where applicable. Another technique used was the reuse of code throughout the program. This is different from redundancy and repetition, as even though the code is similar, it performs a different task and is needed for the requirement for each module to be met. An example of this is shown below.



```
def convert_to_upper(string):
    string = string.upper()
    return string

print(convert_to_upper("i am huzaifa")) #test category 1a

def convert_to_lower(string):
    string = string.lower()
    return string

print(convert_to_lower("I Am Huzaifa")) #test category 1a
```

Figure 3.2: Snippet from Module 1

Yes, there is repetition and redundancy in the program, but any avoidable redundancy was avoided. These modularity concepts are applied throughout the program to make the code run smoothly and for the tasks of each module to be performed without any errors.

### **Review Checklist:**

1. **Has overlapping code been replaced/deleted to reduce repetition?**  
Yes, the code was written as such that modules are not using the exact same code each time; however, some do have similar lines of code performing different tasks
2. **Has code been re-used to perform tasks for different modules?**  
Yes, there is not an extreme amount of repetition in the code, and the code that has been repeated has been reused for similar but not the same tasks.
3. **Are all modules split-up into their respective well-defined tasks?**  
Yes, each module is split up and above each module is a brief explanation as to what the intended task of the module is
4. **Are all modules performing one well-defined task effectively?**  
Yes, each module performs its task without any problems.
5. **Are methods/functions using parameters?**  
Yes
6. **Are methods/functions using return values?**  
Yes
7. **Is each string returned in every module?**  
Yes
8. **At the end of each module, is the string “printed” to indicate that the required task of the module has been implemented when the code is run?**  
Yes, using the print() function
9. **Is each string defined at the start of each module?**  
Yes, using the “def()” function.

**10. Is each string parsed through their respective module?**

Yes

As shown in the checklist above, from the questions that have been derived, there was no refactoring needed to be done, and any issues that came up are shown and logged in the gitlog and version control. The code has been kept simple and concise to avoid any major issues from occurring, and hence no issues arose, and refactoring was not needed for this program.

**Black-box Test Cases:**

**Module 1 (Category 1a):**

Equivalence Testing will be the approach for this module.

Category	Test Data	Expected Result
String is a valid string	"Khan"	KHAN
String is a valid string	"Huzaifa Faisal Khan"	huzaifa faisal khan

**Module 2 (Category 1b):**

Equivalence Testing will be the approach for this module.

Category	Test Data	Expected Result
String contains a numeric value	"I am 20 years old"	There are numeric values in this string
String contains no numeric value	"Pirates of Caribbean"	There are no. numeric values in this string

**Module 3 (Category 1c):**

Equivalence Testing will be the approach for this module.

Category	Test Data	Expected Result
String is an invalid number	"2p67hello"	This is not a valid number
String is a valid number	"8850"	This is a valid number

**Module 4 (Category 1d):**

Equivalence Testing will be the approach for this module.

Category	Test Data	Expected Result
String is a valid string and contains a numeric value	"I am 20 twenty years old"	I AM TWENTY YEARS OLD
String is a valid string and contains a numeric value	"I am 20 twenty years old"	i am twenty years old

### **Module 5 (Category 2c):**

Equivalence Testing will be the approach for this module.

Category	Test Data	Expected Result
String converts hours to minutes	"hours*60"	60 hours is 3600 minutes
String converts minutes to hours	"minutes/60"	60 minutes is 1 hour
String converts minutes to seconds	"minutes*60"	60 minutes is 3600 seconds
String converts seconds to minutes	"seconds/60"	60 seconds is 1 minute

Equivalence testing was the method of approach used to test all the modules. This is because all the modules contained either two certain outcomes, or if and else statements; illustrating whether the content in the string satisfied the desired tasking of the module, or not. This technique was used as it lowers the number of test cases needed for each module, saving time and making the testing method efficient.

### **White-box Test Cases:**

The two modules that white box testing will be implemented on are Module 2 (Category 1b) and Module 3 (Category 1c), as they both contain if and else statements.

Module 2 (Category 1b):

Path	Test Data	Expected Result
Enter "if" if character in string is a numeric value	"I am 20 years old"	There are numeric values in this string
Enter "else" if character in string is not a numeric value	"I am twenty years old"	There are no numeric values in this string

Module 3 (Category 1c):

Path	Test Data	Expected Result
Enter "if" if character in string is a valid number	"8850"	This is a valid number
Enter "else" if character in string is not a valid number	"2p67hello"	This is not a valid number

This method of approach was used to test two of the modules in the program, both of which containing if and else statements, which means that either the outcome is "true" or "false". This is because there are only two outcomes for the code to produce, and white box testing tests all the possible paths in the production. Since there were a only two paths to be take, a table was instructed for each module, indicating the path and what the expected result of the path should be when the test data is run through it. This makes implementing the test code less complicated and ensures efficiency and less errors to occur while coding.

### Test Implementation and Execution:

Module Name	BB Test Design (EP)	WB Test Design	EP Test Code (implemented/run)	BVA test code(implemented/run)	White-Box Testing(implemented/run)
Module 1 (Category 1a)	Implemented	Implemented	Implemented	N/A	Implemented
Module 2 (Category 1b)	Implemented	Implemented	Implemented	N/A	Implemented
Module 3 (Category 1c)	Implemented	Implemented	Implemented	N/A	Implemented
Module 4 (Category 1d)	Implemented	Implemented	Implemented	N/A	Implemented
Module 5 (Category 2d)	Implemented	Implemented	Implemented	N/A	Implemented

The test code was implemented in such a way that all the modules were tested individually within the program, however when the program was run, all the outcomes were illustrated, rather than just one. Black box testing techniques were used for all modules, while white box techniques were used for module 1c and 1d. An example of how testing was done for Module 1 is shown below:

```
1  from productioncode import *
2
3  # Module 1 Category 1a)
4  print("Testing Category 1a) Converting string to upper case:\n")
5  print("Test Data Input: Khan")
6  print("Output: " + convert_to_upper("Khan"))
7  print("\n")
8
9  # Test category 1a
10 print("Testing Category 1a) Converting string to lower case:\n")
11 print("Test Data Input: Huzaifa Faisal Khan")
12 print("Output: " + convert_to_lower("Huzaifa Faisal Khan"))
13 print("\n")
```

Figure 4.1 – Code for testing Module 1

```
Testing Category 1a) Converting string to upper case:
Test Data Input: Khan
Output: KHAN

Testing Category 1a) Converting string to lower case:
Test Data Input: Huzaifa Faisal Khan
Output: huzaifa faisal khan
```

Figure 4.2 – Result of testing Module 1

First, we imported the production code into the test code file so that the test code can run everything in the production code, otherwise we would have no way to test the production code. Print statements are using to clearly show what category is being tested, and what the task of the given module is. To make spaces between each line of the result, “\n” is used, enabling the result of the code to be neater and easily readable. The input that is present is clearly indicated, as well as the output that will be produced by running the code. This clearly shows what is being tested in each module.

## **Version Control:**

Git hub is used to log and commit any changes that were made to the report, the production code, and the test code. The changes are clearly labelled and give a brief description as to what changes are being made, with each change clearly shown in git hub.

[https://github.com/hfkhan13/Khan\\_Huzaifa20188850\\_ISERepo](https://github.com/hfkhan13/Khan_Huzaifa20188850_ISERepo)

## **Ethics and Professionalism:**

a) Ethics and professionalism are a key part of any software engineering project, whether it be a small-scale university project, or a project being conducted by a large software developer company; ethics are vital. Using the code that has been designed and implemented, we can create scenarios as to how the code can be used for a much larger scale project, and hence what harmful effects can be caused if ethics and professionalism are not considered.

For instance, we can take Module 3 (Category 1c) and implement it in a large-scale scenario. The module identifies whether a string contains a valid number or not. An online banking system might use this to identify whether the user has entered a valid number while they were making a payment. When the developers were writing the code, they forgot to write an “else” statement which illustrates that there is no valid number in the string, i.e. the inputted number has been written incorrectly. The system was implemented without this problem being fixed. This could cause many harmful effects to both the user of the system, the company the system was created for and the economy of the country. If the system can’t check whether the user has typed in a valid number, there is a risk that an incorrect amount of money could be paid. This would cause financial harm to the user, as they could be paying more money than they desired to. This could also raise questions to the corporation/individual receiving the money, as they could come under investigation as to why and how they got an abnormal amount of money deposited into their bank account. This could cause further problems such as being framed for something they have not done.

Module 5 (Category 2d) performs the conversion of hours to minutes and vice versa. This can be upscaled and a scenario can be created where this code can be used. For instance, airports could use this method to display when flights will arrive and leave on the big screens. When designing the code, the formula for converting hours to minutes was written incorrectly, and this problem was not picked up while testing. This could cause confusion for passengers and pilots. If the time is not converted correctly, passengers and pilots would arrive at the airport either too late for the flight and/or too early. This would cause major disruption and could cause long delays at the airports. If the airport is a large international airport, this problem could cause aircraft to be grounded, which would cause airline companies and governments to incur financial losses, as the aviation and travel industry are one of the main sources of income for most countries around the world.

We have taken two modules from code and implemented them into real life scenarios to illustrate that if ethics and professionalism are not implemented throughout the course of the project, the effects can be harmful, physically, emotionally, and financially. Therefore, it is important to always follow guidelines and principles for your work to avoid causing harm to yourself and society.

b) Using ACS guidelines, we can identify ways to avoid the problems that have been stated above that could occur due to the lack of ethics and professionalism. The ACS have a list of 6 guidelines and values that an individual should uphold, that are as follows:

1. The Primacy of Public Interest
2. The Enhancement of Quality of Life
3. Honesty
4. Competence
5. Professional Development
6. Professionalism

Using this code of conduct, we can identify two suggestions as to how to avoid ethical problems. Firstly, is it important for the public to be of primary interest. Therefore, there should be numerous testing procedures and cross-examination models before the code can be implemented. Taking into our examples in part a, if correct, rigorous testing methods are implemented by not just the designers of the code, but also by their peers and other individuals, there would be little to no risk for the public to be harmed. Also, while designing the code, the people working on the code should have the public as their primary concern, so all the work they do should be looked through a lens of “would the public be harmed in any way by this piece of work. This can be enforced by the employer by making the employees sign an agreement that public interest and safety is more important than any other personal and financial gain. This means that the workers need to be honest and competent with their work.

Another way to avoid the problems stated in part a is by ensuring that workers go through a background check before being employed and have some sort of academic testing done to make sure that they can complete tasks and projects in an adequate manner. This would make sure that the workers being hired are honest and would not risk their employment and ruining the company’s reputation for their own personal gain. Having honest, competent, and professional employees is integral for a workplace to thrive. If workers are employed just on the basis that they have the qualifications needed for the job, it would open the door to possible misbehaviour and ill-conduct if proper testing and interviews are not implemented to get to know the individual. This would also in hindsight cause public harm and there would be no professional development occurring in the workplace.

The problems that have been identified in the previous section can easily be removed/reduced if proper guidelines are used in the workplace. Using these guidelines is in the best interest of everyone involved, and would improve public opinion, employee satisfaction and the company’s reputation.

### **Discussion:**

This report gives a detailed description on how to apply software engineering codes effectively, while using examples and code to further emphasise the implementation of these techniques. The quality of this work can be improved by better use of vocabulary, and a neater well-presented format. The code could also be a lot more in depth and could be made to a lot more different functions, rather than it being a very simple code. Testing could have been implemented more thoroughly and in a more efficient manner, and the test code also could’ve been much more thorough and in depth as compared to the one in this project. Overall, the report gives the reader a basic understanding and an insight into the world of software engineering, and all its marvels.