

2326504

by Daniel Meskill

Submission date: 02-May-2026 05:19PM (UTC+0100)

Submission ID: 281903023

File name: 52532_Daniel_Meskill_2326504_727104_20824369.docx (890.42K)

Word count: 2838

Character count: 13284

Daniel Meskill

2326504

ACCA4020

Introduction to computer programming

Contents

I**Error! Bookmark not defined.**

ntroduction. 3

Task 2 test cases 5

Task 3 5

Task 2: Correct the Program logic and re-test the corrected version. 7

The Comparison 7

2. Key Features 7

How the specific changes i made resolved the original problems: 8

1. Stopping the "Crash" 8

2. implementing while loop 8

3. Preventing "Data Bleeding" 8

5. Output Formatting 8

Task 3 : Improving logic. 9

A Reflection on the journey. 10

Examples 10

Best Test Case..... 11

Appendix 13

Screenshot a..... 14

Screenshot B..... 14

Introduction.

For this assignment I'm looking to convert a flowchart into python code, this flowchart has been passed on, and contains a few errors.

This code will require information on how many Zones the carpark holds, trying to look at it like disability parking and electric vehicle charging, a third zone could be standard or something like parent and toddler. We also need the information on how many spaces there are within each zone. This code will go through a number of tests to find out where the mistakes are. This will then be corrected

1A

Test case A: The first error was an unresolved reference, under the first "if" Loop there is a line. `total_spaces = Spaces_per_Zone`. This should match the input integer `SpacesPerZone`. Because of this error, it does not allow you to run any of the test examples. Meaning case study A till D will all fail.

However by correcting this, Keeping the underscore on this line, but also adding the underscores to the input command `Spaces_per_zone = int(input( ...` this was done as it makes the content var more readable and tidy

Table 1 – Test cases

Test Case	Zones	Spaces	Input per Zone
A	2	4	Zone 1 : 1, 1, 0, 0 Zone 2 : 0, 0, 1, 1
B	2	3	Zone 1 : 1, 1, 1 Zone 2 : 1, 1, 1
C	1	4	Zone 1: 1,2,0, %
D	2	3	Zone 1 : 1, 0, 1 Zone 2 : 0, 0, 1

Test Case A Link: [Screenshot A](#)

Test Case B Link: [Screenshot B](#)

Test Case C Link: [Screenshot C](#)

Test Case D Link: [Screenshot D](#)

1B,

Here are four test examples of different scenarios, the first errors encountered were the naming conventions. as stated before, this code would not run with the mismatch in place, Another this that had to correct was the loops, this took me a bit of time to figure out, but after looking back on what was learnt and studied in my own time, looking at articles on "Real Python" By Leodanis Ramos (1).

Another apparent error present was the test example D, here it had recurring numbers, this was due to the number's fractional properties, so ideally that should be formatted to around two points after the decimal point.

Later in the code you come across another naming discrepancy, This was again the case when it came to the input commands, but because these are actually displayed, (Printed) it is important to have the space as this gives the ability to read more comfortably.

This for example is the text that is displayed in the programme, "OccupiedStatus".. this should be Occupied_Status. These are not syntax errors so the code can still be run, but this sort of coding is not pleasant to read and can sometimes be referred as dirty code. As discussed here (2)

There was a spelling mistake where on the line of invalid status it was written invalid statis, This was enough to stop the code running but it was something that was easily fixed.

Noticing in the original code there were lines of code in the space counter incremental steps, however this code was not implemented at all like how it could have been, so technically this was counting for no reason, and showed the user no information. this is something that gets further touched on, showing how utilizing into the occupancy status input, gives the user feedback

One of the mistakes found in the flowchart information was that if you inputted any character other than a number, it would crash so a try and except exceptions line was added to stop this crash and inform the user of the invalid character

This flowchart also had code for showing an error when a numerical key was pressed that was not 0 or 1, however once again this was not the case if someone were to input a non numerical key. % for .example

Task 2 test cases

Table 2

Test Case	Observed output	Error type	Explanation
A	Spaces_per_zone	Naming Conventions	The code would not run while this output did not match stored input SpacesPerZone
B	If not while Loops	Logical	An IF command is not a loop, so therefor the Occupancy status question will only run once.
C	Invalid Statis	Typo in output string	If the output stage of occupancy status is wrong it should print invalid status, only here its spelt Statis
D	No space counter	Logic	In the Flowchart there is no space counter being utilized in the occupancy status command
E	ValueError	Logic	There wasn't a Valueerror step to account for a non integer being inputed

Task 3

Test Case	Observed output	Was the issue resolved ?	Explanation
A	Initial error with naming conventions Spaces_per_Zone	Yes	Rather than deleting the underscores from the initial errors. Underscores were added to all of the

			code. This makes reading the code as well as the instructions displayed far easier.
B	If command in the flowchart loop instead of using while	yes	In the Flow chart there were "if" commands when there should have been "while", this is because the zone counter and spaces counter are loops these need to count through the numbers defined at the start.
C	Invalid statis, this is a spelling mistake on the word status	Yes	This was a simple one to fix it was just a quick typo mistake
D	Space counter utilisation	Yes	Originally in the flowchart the space counter was not being used, so this was added it to the occupancy status, this means you can clearly see it counting up as you input occupancy status data
E	ValueError warning	Yes	After a lot of searching it was clear adding "try" "if" "else" and "except" was the best way to implement a ValueError where if a character other than a number was used then a warning would display stating they need to input a 0/1

Task 2: Correct the Program logic and re-test the corrected version.

Screenshot link to: [working task A](#) Screenshot link to: [working task B](#)

Screenshot link to: [working task C](#) Screenshot link to: [working task D](#)

The Comparison

Having now done many iterations of the parking zone monitoring script. The "Initial Model" follows the raw logic of the provided flowchart, while the "Refined Model" introduces advanced coding with commands like "while not". Researched 3)

The transition from the first version to the second is steep. From the initial code that wouldn't even get to the initial first loop

2. Key Features

A. Input Validation & Error Resilience

- Initial Model: this first version relies solely on int(input command, which are quite volatile if someone entered any value other than 0 and one it would cause the system to crash
- Refined Model: this second version has the introduction of try and except commands as well as value errors with taxed like please enter a valid number zero and one instead of crashing.

B. Data Integrity & Loop Precision

- Initial Model: This first version uses an incremental space counter that can easily make mistakes if someone were to add a wrong number
- Refined Model: the program refuses to occupy a parking space until a strictly binary value (0 or 1) is confirmed. This guarantees that the final occupancy percentage is 100% accurate to the zone's capacity.

C. Variable Scoping and Resetting

- Initial Model: Risk of "Accumulation Error." If counters are not strictly reset inside the zone loop, data from Zone A could bleed into Zone B.

- Refined Model: Explicitly resets `occupied_count` and `space_counter` at the start of every outer loop iteration. This treats each parking zone as an independent zone.

The transition from my initial version to the final code was essentially a process of strengthening the "weak points" where a program usually fails.

How the specific changes made resolved the original problems:

1. Stopping the "Crash"

- The Problem: In the first version, the code assumed the user would be perfect. If someone typed "one" instead of "1," or hit a random key, the program would stop so by adding the `try...except` block. This acts as a protective shield. Now, when an error occurs, the "except" section catches it, gives the user a helpful hint, and keeps the program running.

2. implementing while loop

- The Problem: Initial logic often moves to the next parking space even if the input was wrong. This leaves you with data where a zone might say it's 50% full just because the program skipped half the spaces.
- The Resolution: implementing a while loop (`while not valid_status`). This forces the code to stay on the exact same parking space until it gets a valid 0 or 1. It effectively "refuses" to move forward until the data is clean.

3. Preventing "Data Bleeding"

- The Problem: A common issue in the first draft was that the `occupied_count` wasn't being reset properly. If Zone 1 had 5 cars, Zone 2 would start counting at 5 instead of 0.
- The Resolution: moving the initialization of `occupied_count = 0` and `space_counter = 1` inside the outer loop. This ensures that every time a new zone begins, the "slate is wiped clean," and the calculations for that specific zone are independent and accurate.

5. Output Formatting

- The Problem: Raw computer math can be ugly (e.g., 16.666666666666668%).
- The Resolution: Using string formatting (`:.2f`). This doesn't just look better; it makes the human operator who needs to make quick decisions based on the occupancy percentage. Quite recently

So with all that the programme now runs a working code and performs all the functions .required from it.

Task 3 : Improving logic.

To get this code to work it really did take me quite a while to figure out what was required as this couldn't tell from the description of the assignment. Being really unsure what it needed to do outside the basic, at first the Python code was going to be a live system where cars would come in and out of the car park so the counters would count down as well as up. But clearly looking at the stop command this wasn't the case.

Once understood what it was that was asked of me then Sit down look at the code and start pulling out the errors. As mentioned before one of these errors where immediately noticed and that was the loops that should have been while loops but were written down incorrectly. This was so apparent simply because this was learned About loops Quite recently.

Also noticing the naming convention space is free zone was written with underscores but not anywhere else so the two was a quick discovery

The problem was an issue with indentations It took a while to figure this out when I really had to go through it systematically change the indentations one at a time till it was right. A Python code video was used a lot (4) Mosh, this is a YouTube video were a man runs though Python across the span of a couple of hours. This was found to be vey helpful as a top up to my normal lecture.

As Istated previously there were a few difficulties lthat had to be faced when Dealing with the multiple zones especially when implementing 2 loops (while), One of which was zones and the other for spaces also needed to fix the incremental data, in it's initial flow chart this data was not working correctly. also implemented the use of these increments in the input dialogue, so the user could see what space they were dealing with at any given time.

```
one 1, Space 1 - Enter Occupancy status (0/1):
invalid Status, Please do not enter a character, use
one 1, Space 1 - Enter Occupancy status (0/1): 1
one 1, Space 2 - Enter Occupancy status (0/1): 1
one 1, Space 3 - Enter Occupancy status (0/1): 1
one 1, Space 4 - Enter Occupancy status (0/1): 1
one 1, Space 5 - Enter Occupancy status (0/1): 1
one 1, Space 6 - Enter Occupancy status (0/1): 1
one 1, Space 7 - Enter Occupancy status (0/1): 1
one 1, Space 8 - Enter Occupancy status (0/1): 1
one 1, Space 9 - Enter Occupancy status (0/1): 1
one 1, Space 10 - Enter Occupancy status (0/1): |
```

```
input(f'Zone {zone_counter}, Space { } - Enter Occupancy status (0/1): ")
```

Another improvement made, was specifically informing what was expected as a valid occupancy response..

```
print("Invalid Status, Please input a valid status (0/1).")
```

And

```
print("Invalid Status, Please do not enter a character, use (0/1)")
```

A Reflection on the journey.

My journey into programming started a number of years back, this is where I first started writing small programmes and automation script, now while this was just me learning in my own time this was the start of my interest. Unfortunately however I didn't keep it up, This is probably due to the fact that I was not doing this as a career choice but more in my own time and because of this there was no pressure to learn, so when I took this course and started this module, I had to start essentially from the scratch again

If I'm completely honest I found most of the concepts challenging that were in this module, A strong reason for this is due to my medical condition I've been unable to stay in class, missing about an hour every lesson, so I constantly feel like I'm playing catch up. This had to be turned around so I have spent many hours teaching myself through other means, like the very helpful website Real Python and also watching YouTube videos showing for example..what different functions do to your code and how I can implement it in mine.

Examples

To provide some examples of concepts I have applied to the assignment would be:

Try and Except: these are exceptions that I learnt, they can be implemented into your code which works by excepting a possible wrong input from the user (str) not (int) as an example, but instead of crashing, we print invalid input, this way it does not break the while loop.

Placing the {Occupied count} within curly brackets and placing them within the print(f' occupied status line., This offers direct feedback to the user and gives them confirmation of the exact zone they are looking at.

Likewise, to the last example, placing {Zone counter} and {space counter} into the occupied_status input line. This again provides information to the user, showing them what space and what zone they are occupying.

Best Test Case

Looking at the three test cases we did, I believe the third was the best, this is because it was the test case that Ironed out all the creases and was able to fine tune the code whereas the first test case was full of information that I already knew. Example of this would be the naming conventions, because of the mistake, I knew the test would fail furthermore I knew I had to fix the naming conventions as well as implement while loops to get this code to work.

By the time I got to the third test case the mistakes that were found in the flow chart have been corrected which give me a chance and opportunity to write the code in a more effective and practical way.

I believe the only real way to test a product or service or in this case a code is to put it in a real-life scenario. I believe this is the case because you can never be too sure what someone is going to write in your input lines, be it a silly mistake or a deliberate attempt to crash or gain access to the code, Only in a real test case do you actually see all the small coding errors. This is undoubtedly going to change soon with the arrival of artificial intelligence we should not only be able to check your code but I imagine being able to put it into a simulated real life scenario where the AI simulates all the random inputs you would probably receive from a human.

Running these test cases gave me clear understanding that the flowchart, was not to be taken exact and word for word as that implementation would not have worked. Instead, really looking at the flowchart do you see the loop arrows providing you with the information that for this code you must use a while or for loop to enable the code to work, even though in the dialogue boxes its written (if). It has given the understanding that you must use common sense and a level of understanding to know what will work and what will not prior to converting.

I am very pleased with how this code has turned out, I'm also pleased and proud of my ability to convert this flowchart into working code even if it did take me a (while) to do so.

Reference

- (1) Leodanis Ramos, [Python while Loops: Repeating Tasks Conditionally – Real Python](https://realpython.com/python-while-loop/) available at <https://realpython.com/python-while-loop/>, accessed on 4th/April, 2026
- (2) [1] K. Kaya, "Dirty Code: Simple Rules to Avoid It," *DEV Community*, Dec. 18, 2024. (accessed Apr. 22, 2026).
- (3) [2] Real Python, "Nested Loops in Python," *Real Python*, Jan. 13, 2022. (accessed Apr. 22, 2026).
- (4) [3] M. Hamedani. "Programming with Mosh." YouTube. (accessed Apr. 22, 2026).

Appendix

Screenshots A

```
1 numberOfZones = int(input("Input numberOfZones: "))
2 SpacesPerZone = int(input("Input SpacesPerZone: "))
3
4 zone_counter = 1
5 max_free_spaces = -1
6 zone_with_max_free = -1
7
8 while zone_counter <= numberOfZones:
9     space_counter = 1
10    occupied_count = 0
11
12    total_spaces = SpacesPerZone
13
14    while space_counter <= Spaces_Per_Zone:
15
16        valid_status = False
17        while not valid_status:
```

new assignemnt × assignment basic ×

C:\Users\danie\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\danie\PycharmProjec
Input numberOfZones: 2
Input SpacesPerZone: 4
Traceback (most recent call last):
 File "C:\Users\danie\PycharmProjects\PythonProject\assignment_basic.py", line 14, in <module>
 while space_counter <= Spaces_Per_Zone:
 ^^^^^^^^^^^^^^^^^^^^^
NameError: name 'Spaces_Per_Zone' is not defined. Did you mean: 'SpacesPerZone'?

t

Screenshot a

```
1 number_of_zones = int(input("Enter number_of_zones: "))
2 Spaces_Per_Zone = int(input("Enter spaces_per_zone: "))
3
4 zone_counter = 1
5
6 max_free_spaces = -1
7 zone_with_max_free = -1
8
9 while zone_counter <= number_of_zones:
10     space_counter = 1
```

new assignemnt x assignment basic x

```
Enter number_of_zones: 2
Enter spaces_per_zone: 4
Enter status for space 1 (0 for free, 1 for occupied): 1
Enter status for space 2 (0 for free, 1 for occupied): 1
Enter status for space 3 (0 for free, 1 for occupied): 0
Enter status for space 4 (0 for free, 1 for occupied): 0
Zone 1 Results:
- Occupied count: 2
- Occupancy percentage: 50.0%
Enter status for space 1 (0 for free, 1 for occupied): 0
Enter status for space 2 (0 for free, 1 for occupied): 0
Enter status for space 3 (0 for free, 1 for occupied): 1
Enter status for space 4 (0 for free, 1 for occupied): 1
Zone 2 Results:
- Occupied count: 2
- Occupancy percentage: 50.0%
Zone with the most free spaces: 1
Number of free spaces: 2
```

Screenshot B

```
1 number_of_zones = int(input("Enter number_of_zones: "))
2 Spaces_Per_Zone = int(input("Enter spaces_per_zone: "))
3
4 zone_counter = 1
5
6 max_free_spaces = -1
7 zone_with_max_free = -1
8
9 while zone_counter <= number_of_zones:
10     space_counter = 1
```

n new assignemnt x assignment basic x

```
C:\Users\danie\PycharmProjects\PythonProject\.venv\Scripts
Enter number_of_zones: 2
Enter spaces_per_zone: 3
Enter status for space 1 (0 for free, 1 for occupied): 1
Enter status for space 2 (0 for free, 1 for occupied): 1
Enter status for space 3 (0 for free, 1 for occupied): 1
Zone 1 Results:
- Occupied count: 3
- Occupancy percentage: 100.0%
Warning: High Occupancy!
Enter status for space 1 (0 for free, 1 for occupied): 1
Enter status for space 2 (0 for free, 1 for occupied): 1
Enter status for space 3 (0 for free, 1 for occupied): 1
Zone 2 Results:
- Occupied count: 3
```

Screenshot C

```
Enter number_of_zones: 1
Enter spaces_per_zone: 4
Enter status for space 1 (0 for free, 1 for occupied): 1
Enter status for space 2 (0 for free, 1 for occupied): 2
Invalid Status. Please enter 0 or 1.
Enter status for space 2 (0 for free, 1 for occupied): 0
Enter status for space 3 (0 for free, 1 for occupied): %
Invalid input. Please enter a number.
Enter status for space 3 (0 for free, 1 for occupied):
```

Screenshot D

```
Enter number_of_zones: 2
Enter spaces_per_zone: 3
Enter status for space 1 (0 for free, 1 for occupied): 1
Enter status for space 2 (0 for free, 1 for occupied): 0
Enter status for space 3 (0 for free, 1 for occupied): 1
Zone 1 Results:
- Occupied count: 2
- Occupancy percentage: 66.66666666666666%
Enter status for space 1 (0 for free, 1 for occupied): 0
Enter status for space 2 (0 for free, 1 for occupied): 0
Enter status for space 3 (0 for free, 1 for occupied): 1
Zone 2 Results:
- Occupied count: 1
- Occupancy percentage: 33.33333333333333%
Zone with the most free spaces: 2
Number of free spaces: 2

Process finished with exit code 0
```

Working task A

```
Input number Of Zones: 2
Input Spaces Per Zone: 4
Zone 1, Space 1 - Enter Occupancy status (0/1): 1
Zone 1, Space 2 - Enter Occupancy status (0/1): 1
Zone 1, Space 3 - Enter Occupancy status (0/1): 0
Zone 1, Space 4 - Enter Occupancy status (0/1): 0
Occupied count: 2, Occupancy percentage of the zone: 50.00%
Zone 2, Space 1 - Enter Occupancy status (0/1): 0
Zone 2, Space 2 - Enter Occupancy status (0/1): 0
Zone 2, Space 3 - Enter Occupancy status (0/1): 1
Zone 2, Space 4 - Enter Occupancy status (0/1): 1
Occupied count: 2, Occupancy percentage of the zone: 50.00%
Zone with the most free spaces: 1
Number of free spaces: 2

Process finished with exit code 0
```

Working task B

```
C:\Users\danie\PycharmProjects\PythonProject\.venv\Scripts\p
Input number Of Zones: 2
Input Spaces Per Zone: 3
Zone 1, Space 1 - Enter Occupancy status (0/1): 1
Zone 1, Space 2 - Enter Occupancy status (0/1): 1
Zone 1, Space 3 - Enter Occupancy status (0/1): 1
Occupied count: 3, Occupancy percentage of the zone: 100.00%
Warning: High Occupancy!
Zone 2, Space 1 - Enter Occupancy status (0/1): 1
Zone 2, Space 2 - Enter Occupancy status (0/1): 1
Zone 2, Space 3 - Enter Occupancy status (0/1): 1
Occupied count: 3, Occupancy percentage of the zone: 100.00%
Warning: High Occupancy!
Zone with the most free spaces: 1
Number of free spaces: 0

Process finished with exit code 0
```

Working Task C

```
Input number Of Zones: 1
Input Spaces Per Zone: 4
Zone 1, Space 1 - Enter Occupancy status (0/1): 1
Zone 1, Space 2 - Enter Occupancy status (0/1): 2
Invalid Status, Please input a valid status (0/1).
Zone 1, Space 2 - Enter Occupancy status (0/1): 0
Zone 1, Space 3 - Enter Occupancy status (0/1): %
Invalid Status, Please do not enter a character, use (0/1)
Zone 1, Space 3 - Enter Occupancy status (0/1):
```

Working Task D

```
Input number Of Zones: 2
Input Spaces Per Zone: 3
Zone 1, Space 1 - Enter Occupancy status (0/1): 1
Zone 1, Space 2 - Enter Occupancy status (0/1): 0
Zone 1, Space 3 - Enter Occupancy status (0/1): 1
Occupied count: 2, Occupancy percentage of the zone: 66.67%
Zone 2, Space 1 - Enter Occupancy status (0/1): 0
Zone 2, Space 2 - Enter Occupancy status (0/1): 0
Zone 2, Space 3 - Enter Occupancy status (0/1): 1
Occupied count: 1, Occupancy percentage of the zone: 33.33%
Zone with the most free spaces: 2
Number of free spaces: 2

Process finished with exit code 0
```

The Code

```
1 number_Of_Zones = int(input("Input number Of Zones: "))
2 Spaces_Per_Zone = int(input("Input Spaces Per Zone: "))
3
4 zone_counter = 1
5 max_free_spaces = -1
6 zone_with_max_free = -1
7 # These are the first and second loops, ritten as IF in the flowchart, which is wrong
8 while zone_counter <= number_Of_Zones:
9     # Incrementals
10    space_counter = 1
11    occupied_count = 0
12    total_spaces = Spaces_Per_Zone
13
14    while space_counter <= Spaces_Per_Zone:
15        valid_status = False
16        while not valid_status:
17            try: # ths stops the code fro crashing and forces the user to input the correct character
18                # occupancy status input
19                Occupancy_Status = int(
20                    input(f"Zone {zone_counter}, Space {space_counter} - Enter Occupancy status (0/1): "))
21
22                if Occupancy_Status == 0 or Occupancy_Status == 1:
23                    # this checks the answer is valid
24                    valid_status = True
25                else:
26                    print("Invalid Status, Please input a valid status (0/1).")
27
28            except ValueError:
29                print("Invalid Status, Please do not enter a character, use (0/1)")
30
31        # incrementals
32        if Occupancy_Status == 1:
33            occupied_count += 1
34
35        space_counter += 1
36
37    free_spaces = total_spaces - occupied_count
38    occupancy_percent = (occupied_count / total_spaces) * 100
39    # this is formatted so the answer only displays 2 decimel places, and stops reaccurences
40    print(f"Occupied count: {occupied_count}, Occupancy percentage of the zone: {occupancy_percent:.2f}%")
41
42    if occupancy_percent <= 80:
43        print("Warning: High Occupancy!")
44
45    if free_spaces > max_free_spaces:
46        max_free_spaces = free_spaces
47        zone_with_max_free = zone_counter
48
49    zone_counter += 1
50
51    print(f"Zone with the most free spaces: {zone_with_max_free}")
52    print(f"Number of free spaces: {max_free_spaces}")
```

FINAL GRADE

GENERAL COMMENTS

89 / 100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

DANIEL MESKILL**89****TASK 1A [20]**

Correct implementation of flowchart (even with errors) [8] Code runs (even if incorrect output) [5] All test cases attempted [4] Screenshots/evidence provided [3]



Flowchart logic has been implemented correctly as provided. All test cases executed with clear evidence
All relevant errors are identified and reported

20**TASK 1B [15]**

Correct outputs recorded [5] Identification of error types [5] Clear explanation of errors [5]



Most of the errors were identified, but important issues such as equal-zone logic, is not well explained.

12**TASK 2A [20]**

Errors correctly fixed [10] Improved logic (loops, conditions) [6] Input validation added [4]



All major errors resolved; logic is clear and robust with good validation. However, equal-zone logic is not well reported

17

TASK 2B [20]

All test cases re-run [6] Outputs clearly recorded [6] Comparison with old results [4] Explanation of fixes [4]



All test cases re-tested with accurate outputs and clear comparison showing improvements.

20

TASK 3 [10]

Understanding of difficulty [5] Impact on design/efficiency [5]



Good explanation with some insight into design challenges

7

TASK 4 [15]

Identifies challenges [5] Examples from assignment [5] Reflection on testing/learning [5]



Good reflection with relevant examples

13