

Laurel Black Assignment

by Laurel Black

Submission date: 18-May-2026 06:54PM (UTC+0100)

Submission ID: 283967116

File name: 66122_Laurel_Black_Laurel_Black_Assignment_727104_378253242.docx (4.21M)

Word count: 3066

Character count: 15802



Programming

Laurel Black 2517390

**Introduction to Computer Programming
ACCA4020**

University of Wales Trinity Saint David

Submitted 18th May 2026

Abstract

Digital innovation now plays an essential role within society, including academic settings such as a university. This report includes a proposed concept of a smart parking lot management system within a university. Originally presented with the use of a flowchart, to be implemented for the computer programming language known as Python, alongside the potential of errors to be found within the flowchart. Test cases were performed and offered insights to an improved and optimised program.

Contents

1. Introduction	1
2. Flowchart Code	3
2.1 Exact logic, Table Test Cases	4
2.2 Results	5
2.3. Identifying errors.....	7
3. Test cases from Flowchart, output observations	10
3.1 Corrections and Revisit	12
3.2 Re-test.....	13
Table 4: Re-test.....	15
5. Improving Logic.....	18
6. Reflection and conclusion	20
7. References	24

List of Figures

Figure 1 - Flowchart	2
Figure 2 - Flowchart Code.....	3
Figure 3 - Test Case A	5
Figure 4 - Test Case B	5
Figure 5 - Test Case C	5
Figure 6 - Test Case D	5
Figure 7 - Syntax error from Flowchart implementation.....	7
Figure 8 - syntax error from Flowchart implementation	7
Figure 9 - Code from Flowchart with invalid syntax.....	8
Figure 10 - Code from Flowchart syntax error.....	9
Figure 11 - Re-Test Cases	12
Figure 12 - Re-Test Case A	13
Figure 13 - Re-Test Case B	13
Figure 14 - Re-Test Case C	13
Figure 15 - Re-Test Case D	13
Figure 16 - Code from The Farmer Was Replaced, in particular use of the “or” operator.....	21
Figure 17 - The Farmer Was Replaced	22

List of Tables

Table 1 - Test Case Formatting.....	4
Table 2 - Output observations	10
Table 3 - Re-test	15

1. Introduction

The Flowchart which centres the programming task refers to using the programming language Python, which will be discussed throughout this document.

The IDE, or fully known as Integrated Development Environment, PyCharm was chosen in terms of writing the Python code. PyCharm is an IDE provided by JetBrains. PyCharm was primarily chosen as the IDE of choice for being out of the box and providing a simple to use experience [1]. No other IDE or text editor was considered for this task.

Logical flaws and errors were immediately identified within the proposed flowchart and later examined within this document. The challenges these presented will also be discussed.

Ensuring uniformity in code structure is imperative for clarity and use in concerns to the smart parking system, in conclusion resulting in careful analysis and test cases.

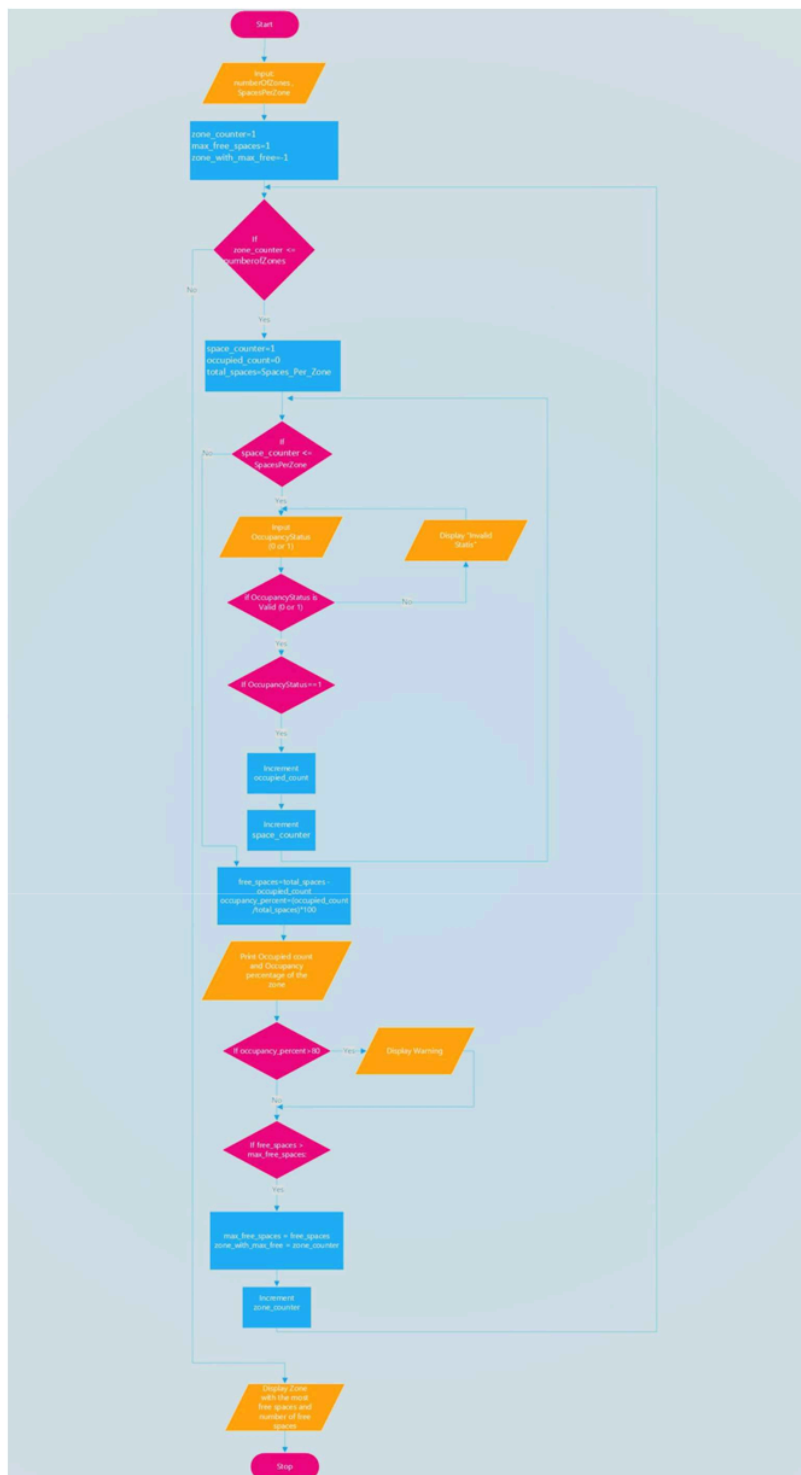


Figure 1 - Flowchart

2. Flowchart Code

```
1  numberOfZones = int(input("Enter number Of Zones: "))
2  SpacesPerZone = int(input("Enter number Of Spaces Per Zone: "))
3
4  zone_counter=1
5  max_free_spaces=1
6  zone_with_max_free_spaces=-1
7
8  while zone_counter <= numberOfZones:
9      space_counter=1
10     occupied_count=0
11     total_spaces=SpacesPerZone
12
13     while space_counter <= SpacesPerZone:
14         occupancy_status = int(input(f"Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): "))
15
16         if occupancy_status == 0 or occupancy_status == 1:
17             if occupancy_status == 1:
18                 occupied_count += 1
19                 space_counter += 1
20             else:
21                 print("Invalid Status")
22
23         free_spaces=total_spaces-occupied_count
24         occupancy_percent=(occupied_count/total_spaces)*100
25
26     print(f"Zone {zone_counter}: {occupied_count} and {occupancy_percent}% of the zone")
27
28     if occupancy_percent > 80:
29         print(f"Warning")
30
31     if free_spaces > max_free_spaces:
32         max_free_spaces = free_spaces
33         zone_with_max_free_spaces = zone_counter
34
35     zone_counter += 1
36
37 print(f"Display zone with the most free spaces: {zone_with_max_free_spaces} and {max_free_spaces} free spaces")
```

Figure 2 - Flowchart Code

Concerning the pseudocode derived from the flowchart, the stages labelled as “Start” and “Stop” represent the initiation and termination of the program and are not translated to python code.

The flowchart’s rectangles signify processes or calculations, such as counting occupied spaces or calculating the occupancy percentage.

Each rectangle represents a discrete step in the program, where data is evaluated, which can include arithmetic operations or statements.

Parallelograms in the flowchart represent input or output operations.

Diamonds signify decision points within the program, requiring conditional evaluation, including if, elif and else statements in Python; allowing the program to

make logical choices based on the user's input or calculated values. Primary examples in this program involve decisions such as parking space occupation, resulting in a 0 or 1 from the user, and whether the occupancy percent exceeds the specified thresholds.

Arrows illustrate the flow of the program, representing the order of which decisions occur.

2.1 Exact logic, Table Test Cases

Prior to executing the test cases, conclusions were met regarding minor flaws within the flowchart. These suspicions were based on inconsistencies observed during the initial speculation of the flowchart, such as unclear conditions for occupancy warnings or potential miscalculations of free spaces.

Running the code with the Test Case scenarios allowed empirical testing, revealing errors that were not proven upon mere observation of the flowchart.

The Test Cases provide the opportunity to validate the logic; observing the program's behaviour provides essential insights for error handling and optimising the code. It is also imperative that accurate translation of the flowchart is understood to result in executable Python statements.

Table 1 - Test Case Formatting

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

2.2 Results

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py"
Enter number Of Zones: 2
Enter number Of Spaces Per Zone: 4
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 1
Zone 1, Space 3 status (0 free, 1 occupied): 0
Zone 1, Space 4 status (0 free, 1 occupied): 0
Zone 1: 2 and 50.0% of the zone
Zone 2, Space 1 status (0 free, 1 occupied): 0
Zone 2, Space 2 status (0 free, 1 occupied): 0
Zone 2, Space 3 status (0 free, 1 occupied): 1
Zone 2, Space 4 status (0 free, 1 occupied): 1
Zone 2: 2 and 50.0% of the zone
Display zone with the most free spaces 1 and 2 free spaces

Process finished with exit code 0
```

Figure 3 - Test Case A

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py"
Enter number Of Zones: 2
Enter number Of Spaces Per Zone: 3
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 1
Zone 1, Space 3 status (0 free, 1 occupied): 1
Zone 1: 3 and 100.0% of the zone
Warning
Zone 2, Space 1 status (0 free, 1 occupied): 1
Zone 2, Space 2 status (0 free, 1 occupied): 1
Zone 2, Space 3 status (0 free, 1 occupied): 1
Zone 2: 3 and 100.0% of the zone
Warning
Display zone with the most free spaces: -1 and 1 free spaces

Process finished with exit code 0
```

Figure 4 - Test Case B

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py"
Enter number Of Zones: 1
Enter number Of Spaces Per Zone: 4
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 2
Invalid Status
Zone 1, Space 2 status (0 free, 1 occupied): 0
Zone 1, Space 3 status (0 free, 1 occupied): %
Traceback (most recent call last):
  File "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py", line 14, in <module>
    occupancy_status = int(input(f"Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): "))
ValueError: invalid literal for int() with base 10: '%'

Process finished with exit code 1
```

Figure 5 - Test Case C

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py"
Enter number Of Zones: 2
Enter number Of Spaces Per Zone: 3
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 0
Zone 1, Space 3 status (0 free, 1 occupied): 1
Zone 1: 2 and 66.66666666666666% of the zone
Zone 2, Space 1 status (0 free, 1 occupied): 0
Zone 2, Space 2 status (0 free, 1 occupied): 0
Zone 2, Space 3 status (0 free, 1 occupied): 1
Zone 2: 1 and 33.33333333333333% of the zone
Display zone with the most free spaces: 2 and 2 free spaces

Process finished with exit code 0
```

Figure 6 - Test Case D

Each Test Case in Figures 3 to 6 shows how the system responds to different input scenarios. Observing the outputs in these figures allows for a clear comparison between the flowchart's intended design and the program performance, resulting in a

valid foundation for error identification and preparing for corrections.

2.3. Identifying errors

```
1  numberOfZones = int(input("Enter number Of Zones: "))
2  SpacesPerZone = int(input("Enter number Of Spaces Per Zone: "))
3
4  zone_counter=1
5  max_free_spaces=1
6  zone_with_max_free_spaces=-1
7
8  while zone_counter <= numberOfZones:
9      space_counter=1
10     occupied_count=0
11     total_spaces=SpacesPerZone
12
13     while space_counter <= SpacesPerZone:
14         occupancy_status = int(input(f"Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): "))
15
16         if occupancy_status == 0 or occupancy_status == 1:
17             if occupancy_status == 1:
18                 occupied_count += 1
19                 space_counter += 1
20             else:
21                 print("Invalid Statis")
22
23         free_spaces=total_spaces-occupied_count
24         occupancy_percent=(occupied_count/total_spaces)*100
25
26         print(f"Zone {zone_counter}: {free_spaces} and {occupancy_percent}% of the zone")
27
28         If occupancy_percent > 80:
29             print(f"Warning")
30
31         if free_spaces > max_free_spaces:
32             max_free_spaces = free_spaces
33             zone_with_max_free_spaces = zone_counter
34
35         zone_counter += 1
36
37     print(f"Display zone with the most free spaces {zone_with_max_free_spaces} and {max_free_spaces} free spaces")
```

Figure 7 - Syntax error from Flowchart implementation

```
C:\Users\Laurel\PycharmProjects\PythonProject\venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py"
File "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py", line 28
    If occupancy_percent > 80:
    ^^^^^^^^^^^^^^^^^^^^^
SyntaxError: invalid syntax

Process finished with exit code 1
```

Figure 8 - syntax error from Flowchart implementation

In addition to the original code from the flowchart, a minor formatting error was found and tested; the original flowchart shows a lack of uniformity in communicating the use for python, in particular concerning If occupancy_percent > 80:, as shown in Figure 7. Further discussions regarding this will be addressed in 5. Reflection and conclusion.

Python will not accept a capitalisation of the 'if' statement, which will immediately terminate the program from successfully running, as shown in Figure 8. This would be the outcome regardless of the Test Case scenario.

Syntax errors will prevent python from running code. A couple of other examples that would be included to trigger a syntax error would be missing colons or poor indentation [2].

Uniformity for clear communication for the programmer is recommended to understand the purpose and use, in this case being for the university smart parking system.

Correcting errors such as a syntax error is relatively simple to resolve, as previously demonstrated. It should be considered that errors such as semantic errors would require code examination [3].

```
1  numberOfZones = int(input("Enter number Of Zones: "))
2  SpacesPerZone = int(input("Enter number Of Spaces Per Zone: "))
3
4  zone_counter=1
5  max_free_spaces=1
6  zone_with_max_free_spaces=-1
7
8  while zone_counter <= numberOfZones_
9      space_counter=1
10     occupied_count=0
11     total_spaces=SpacesPerZone
12
13     while space_counter <= SpacesPerZone_
14         occupancy_status = int(input(f"Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): "))
15
16         if occupancy_status == 0 or occupancy_status == 1_
17             if occupancy_status == 1:
18                 occupied_count += 1
19                 space_counter += 1
20             else:
21                 print("Invalid Statis")
22
23         free_spaces=total_spaces-occupied_count
24         occupancy_percent=(occupied_count/total_spaces)*100
25
26         print(f"Zone {zone_counter}: {free_spaces} and {occupancy_percent}% of the zone")
27
28         If occupancy_percent > 80:_
29             print(f"Warning")
30
31         if free_spaces > max_free_spaces:
32             max_free_spaces = free_spaces
33             zone_with_max_free_spaces = zone_counter
34
35         zone_counter += 1_
36
37     print(f"Display zone with the most free spaces {zone_with_max_free_spaces} and {max_free_spaces} free spaces")
```

Figure 9 - Code from Flowchart with invalid syntax

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py"
File "C:\Users\Laurel\PycharmProjects\PythonProject\main main assignment from flowchart.py", line 28
    if occupancy_percent > 80:
    ^^^^^^^^^^^^^^^^^^^^^
SyntaxError: invalid syntax

Process finished with exit code 1
```

Figure 10 - Code from Flowchart syntax error

3. Test cases from Flowchart, output observations

Table 2 - Output observations

Test Case	Observed output	Error Type	Explanation
Test Case A, Figure 3	Zone 1: 2 and 50.0% of the zone Zone 2: 2 and 50.0% of the zone Display zone with the most free spaces 1 and 2 free spaces	No errors in program	Program logic executed correctly. Minor improvements in output formatting recommended.
Test Case B, Figure 4	Zone 1: 3 and 100% of the zone Warning Zone 2: 0 and 0.0% of the zone Display zone with the most free spaces: 2 and 3 free spaces	Minor logical error. This test case should represent a full university smart car park system. max_free_spaces = 1 For the parking system this needs to be 0. Set too high for the possible result in the test case	100% of the zone is not clear for intended university smart car park system use. Warning is irrelevant due to parking being full occupancy. Spaces should not be full. Misleading.
Test Case C Figure 5	Invalid Statis Line 14, in <module> occupancy_status = int(input(f'Zone {zone_counter}, Space	ValueError: invalid literal for int() with base 10: '%' Input error.	'Invalid Statis' Does not clearly inform the user that a 0 or 1 is mandatory to communicate if a space is free or

Test Case	Observed output	Error Type	Explanation
	{space_counter) status (0 free, 1 occupied): “))		occupied. The code does not permit alternative user inputs other than 0 or 1 in handling the string.
Test Case D Figure 6	Zone 1: 2 and 66.66666666666666% of the zone Zone 2: 1 and 33.33333333333333% of the zone Display zone with the most free spaces: 2 and 2 free spaces	No errors in programme	While not a mathematical or logical error, readability could be improved for the formatting. Slight improvements to wording for the final print could be improved for clarity.

3.1 Corrections and Revisit

```
1  # Get the number of zones and spaces per zone from the console user
2  numberofzones = int(input("Enter number of Zones: "))
3  spacesperzone = int(input("Enter number of Spaces per Zone: "))
4
5  # Variables
6  zone_counter = 1
7  max_free_spaces = 0
8  zone_with_max_free = -1 # This will hold the zone number with the most free spaces
9
10 # Main while loop to process zones
11 while zone_counter <= numberofzones:
12     space_counter = 1
13     occupied_count = 0 # Reset the occupied count for each zone
14
15     # Inner loop to check occupancy of each space in the zone
16     while space_counter <= spacesperzone:
17         occupancy_status = int(input(f"Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): "))
18
19         # Only accepting 0 or 1
20         if occupancy_status == 0 or occupancy_status == 1:
21             if occupancy_status == 1:
22                 occupied_count += 1
23             space_counter += 1
24         else:
25             print("Invalid status, please enter 0 or 1.") # Show error message if invalid input
26
27     # Calculate total spaces, free spaces, and occupancy percentage for the zone
28     total_spaces = spacesperzone
29     free_spaces = total_spaces - occupied_count
30     occupancy_percent = (occupied_count / total_spaces) * 100
31
32     # Print occupancy zone details
33     print(f"Zone {zone_counter}: {free_spaces} free spaces, {occupancy_percent:.2f}% occupied.")
34
35     if occupancy_percent > 80 and occupancy_percent < 100:
36         print(f"Warning: Zone {zone_counter} is nearly full! Over 80% full.")
37
38     # Track the zone with the most free spaces
39     if free_spaces > max_free_spaces:
40         max_free_spaces = free_spaces
41         zone_with_max_free = zone_counter
42
43     zone_counter += 1 # Move to the next zone
44
45 # Output the result based on the free spaces found
46 if zone_with_max_free == -1:
47     print("All zones full.")
48 else:
49     print(f"Display zone with the most free spaces: Zone {zone_with_max_free} ({max_free_spaces} free spaces)")
```

Figure 11 - Re-Test Cases

3.2 Re-test

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\assignment final snake case.py"
Enter number of Zones: 2
Enter number of Spaces per Zone: 4
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 1
Zone 1, Space 3 status (0 free, 1 occupied): 0
Zone 1, Space 4 status (0 free, 1 occupied): 0
Zone 1: 2 free spaces, 50.00% occupied.
Zone 2, Space 1 status (0 free, 1 occupied): 0
Zone 2, Space 2 status (0 free, 1 occupied): 0
Zone 2, Space 3 status (0 free, 1 occupied): 1
Zone 2, Space 4 status (0 free, 1 occupied): 1
Zone 2: 2 free spaces, 50.00% occupied.
Display zone with the most free spaces: Zone 1 (2 free spaces)
```

Figure 12 - Re-Test Case A

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\assignment final snake case.py"
Enter number of Zones: 2
Enter number of Spaces per Zone: 3
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 1
Zone 1, Space 3 status (0 free, 1 occupied): 1
Zone 1: 0 free spaces, 100.00% occupied.
Zone 2, Space 1 status (0 free, 1 occupied): 1
Zone 2, Space 2 status (0 free, 1 occupied): 1
Zone 2, Space 3 status (0 free, 1 occupied): 1
Zone 2: 0 free spaces, 100.00% occupied.
All zones full.
Process finished with exit code 0
```

Figure 13 - Re-Test Case B

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\assignment final snake case.py"
Enter number of Zones: 1
Enter number of Spaces per Zone: 4
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 2
Invalid status, please enter 0 or 1.
Zone 1, Space 2 status (0 free, 1 occupied): 0
Zone 1, Space 3 status (0 free, 1 occupied): %
Traceback (most recent call last):
  File "C:\Users\Laurel\PycharmProjects\PythonProject\assignment final snake case.py", line 17, in <module>
    occupancy_status = int(input(f'Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): '))
ValueError: invalid literal for int() with base 10: '%'
Process finished with exit code 1
```

Figure 14 - Re-Test Case C

```
C:\Users\Laurel\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\Laurel\PycharmProjects\PythonProject\assignment final snake case.py"
Enter number of Zones: 2
Enter number of Spaces per Zone: 3
Zone 1, Space 1 status (0 free, 1 occupied): 1
Zone 1, Space 2 status (0 free, 1 occupied): 0
Zone 1, Space 3 status (0 free, 1 occupied): 1
Zone 1: 1 free spaces, 66.67% occupied.
Zone 2, Space 1 status (0 free, 1 occupied): 0
Zone 2, Space 2 status (0 free, 1 occupied): 0
Zone 2, Space 3 status (0 free, 1 occupied): 1
Zone 2: 2 free spaces, 33.33% occupied.
Display zone with the most free spaces: Zone 2 (2 free spaces)
Process finished with exit code 0
```

Figure 15 - Re-Test Case D

As shown in Re-Test figures 12 to 15, each Re-Test demonstrates improved handling of occupancy calculations and user communications. In contrast to Test

Case Figures 3 to 6, improvements in usability and logic are ensured, resulting in a more reliable program.

Table 4: Re-test

Table 3 - Re-test

Test Case	Observed Output	Was the issue resolved?	Explanation
Re-Test Case A, Figure 12	Zone 1: 2 free spaces, 50.00% occupied. Zone 2: 2 free spaces, 50.00% occupied. Display zone with the most free spaces: Zone 1 (2 free spaces)	Clearer communication regarding free spaces. Little to no clarity for output	Optimised code uses a string to bring the output to two decimal places .2f
Re-Test Case B, Figure 13	Zone 1: 0 free spaces, 100.00% occupied. Zone 2: 0 free spaces, 100.00% occupied. All zones full.	max_free_spaces=0 the condition free_spaces > max_free_spaces is never true in python for the original code. The original code had bad values	Communicating more clearly to the user that there are 0 free spaces in either Zone 1 nor Zone 2. The Warning trigger from over 80% has been removed as was not deemed necessary for a full car park, if occupancy_percent > 80 and occupancy_percent < 100: print(f"Warning: Zone

Test Case	Observed Output	Was the issue resolved?	Explanation
			{zone_counter} is nearly full! Over 80% full.") In this event, if zone_with_max_free == -1: print("All zones full.") has been included.
Re-Test Case C, Figure 14	Invalid Status, please enter 0 or 1. Line 14, in <module> <pre> occupancy_status = int(input(f"Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): "))occupancy_status = int(input(f"Zone {zone_counter}, Space {space_counter} status (0 free, 1 occupied): ")) </pre>	ValueError: invalid literal for int() with base 10: '%' This version of the code does not include conditions that permit alternatives to integers in the strings and will not work unless an integer is handled.	Clearer communications have been resolved for when the incorrect integer has been input. Exit conditions could be considered for the loop if the user accidentally inputs non-integer text.

Test Case	Observed Output	Was the issue resolved?	Explanation
	ValueError: invalid literal for int() with base 10: '%'		
Re-Test Case C, Figure 15	Zone 1: 1 free spaces, 66.67% occupied. Zone 2: 2 free spaces, 33.33% occupied. Display zone with the most free spaces: Zone 2 (2 free spaces)	{occupancy_percent:.2f}	The string method of .2f performs better readability and use of the program, resulting in it outputting the occupied count to 2 decimal places. Writing has been improved for better clarity on the final print for use.

5. Improving Logic

Concerning good practices in terms of the program, it is preferable for the syntax to be uniform in the code. The improved code has been changed to operate using snake case in entirety; resulting in the variables being set in lower case. Snake case is generally used for Python [2].

In this program, variables are used to track critical information for each zone and space, including (zone_counter, space_counter, and occupied_count).

In Python, variables are assigned with the operator (=), and there is no need to declare their type before use. Unlike other programming languages such as C, Python variables are labels that reference objects, which could include numbers, strings, lists, or other types [4]. The versatility of Python allows values to be stored and updated dynamically while the program is running.

Comments have been included throughout the code to describe the purpose of each variable as well as the flow of logic, enhancing readability for a better understanding of the code. Comments in Python are used with the (#) symbol, which Python will ignore during the running of the code. Comments are excellent for notes to explain steps within the code for both the programmers creating the code, or other programmers reading the code [2].

Nested loops are used to handle zones and spaces. The outer loop iterates each parking zone, while the inner loop iterates over each parking space within the zone; using a nested loop. Each iteration of the loop triggers a full iteration, allowing the program to process each space individually.

Nested loops can become more complex with extra layers [2]. It is imperative to keep the variable names and structure clear to maintain readability.

As shown in the Test Case Figures D, 6 and 13, the original code implemented from the flowchart does not include a string to bring the result to a more readable result for human use, such as bringing the result to two decimal places, as much this is still mathematically logical and correct. As shown in the final code in Figure 9, the string

{occupancy_percent:.2f}) was included.

Formatting strings, or F-Strings, were introduced in Python 3.6, with previous versions requiring use of the format() method [5].

In the final code as shown in Re-Test Case in Figure 11, the placeholder is used ({}) to store the occupancy_percent and use of the {:.2f} string to display the result with 2 decimal places

Both versions of the code encountered a common difficulty when the user entered an invalid input, such as (&) or (%), resulting in the (ValueError: invalid literal for int() with base 10), as shown in Test Case Figures C, 8 and 12.

In these cases, the program expects an integer, but the input does not represent a whole number in base 10, and therefore cannot be converted successfully. Handling this issue was challenging due to the programming processing multiple zones and spaces using nested loops. Each iteration relies on manually tracked variables; invalid input can disrupt the flow of the loops and cause calculation issues regarding the occupancies and free spaces. The limitation remained within the programs final design as shown in Figure 11 to ensure the loops within the program could continue operating correctly.

While maintaining this approach in the Re-Test case maximised efficiency by having a reduced amount of data structures, it continues to limit user input in situations such as human error.

Several solutions could be implemented to resolve this problem, such as exception handling (try) or (except), could be applied to result in a more robust program from terminating from user errors [6].

6. Reflection and conclusion

Several challenges were identified for the programmer and the program throughout the problem solving and optimisation process.

During the process of implementing the code directly from the flowchart, a lack of intuition was experienced on the representation of “If occupancy_percent > 80”, as addressed in 1b, Identifying Errors, shown in Figures 1 and 2.

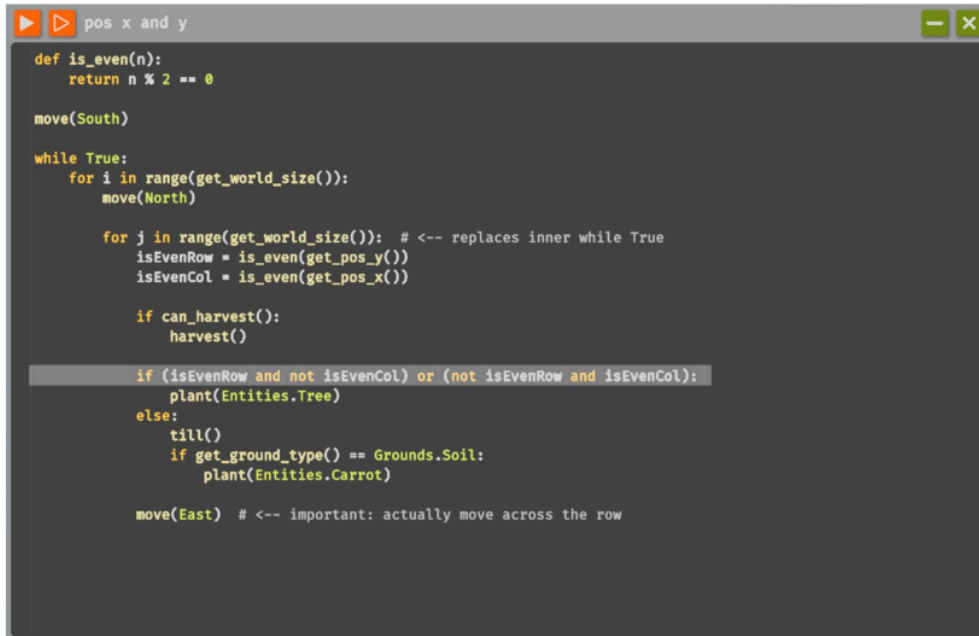
The primary cause of this causing a detriment to implementing the flowchart into Python code was a personal interpretation error and frustration in reading the flowchart.

As shown in the flowchart in Figure 1, the flowchart typically shows uniformity in correct use of the (if/lf) statements, barring this scenario. It is advisable to maintain uniformity and clarity when referring to flowcharts and pseudocode.

Compared to languages such as C, Python is a high-level programming language with dynamic semantics [7]. Experience with C in contrast in which the most common logical operators are (&&) for local and, (||) for logical or and (!) for logical not [8]. not caused some delays in grasping a fundamental use of Python.

Solutions to grasping this language was found in understanding Python in a Python inspired video game shown below, known as The Farmer Was Replaced. The

Farmer Was Replaced uses a custom interpreter with syntax highly akin the programming language Python [9].



```
pos x and y

def is_even(n):
    return n % 2 == 0

move(South)

while True:
    for i in range(get_world_size()):
        move(North)

        for j in range(get_world_size()): # <-- replaces inner while True
            isEvenRow = is_even(get_pos_y())
            isEvenCol = is_even(get_pos_x())

            if can_harvest():
                harvest()

            if (isEvenRow and not isEvenCol) or (not isEvenRow and isEvenCol):
                plant(Entities.Tree)
            else:
                till()
                if get_ground_type() == Grounds.Soil:
                    plant(Entities.Carrot)

            move(East) # <-- important: actually move across the row
```

Figure 16 - Code from *The Farmer Was Replaced*, in particular use of the “or” operator.

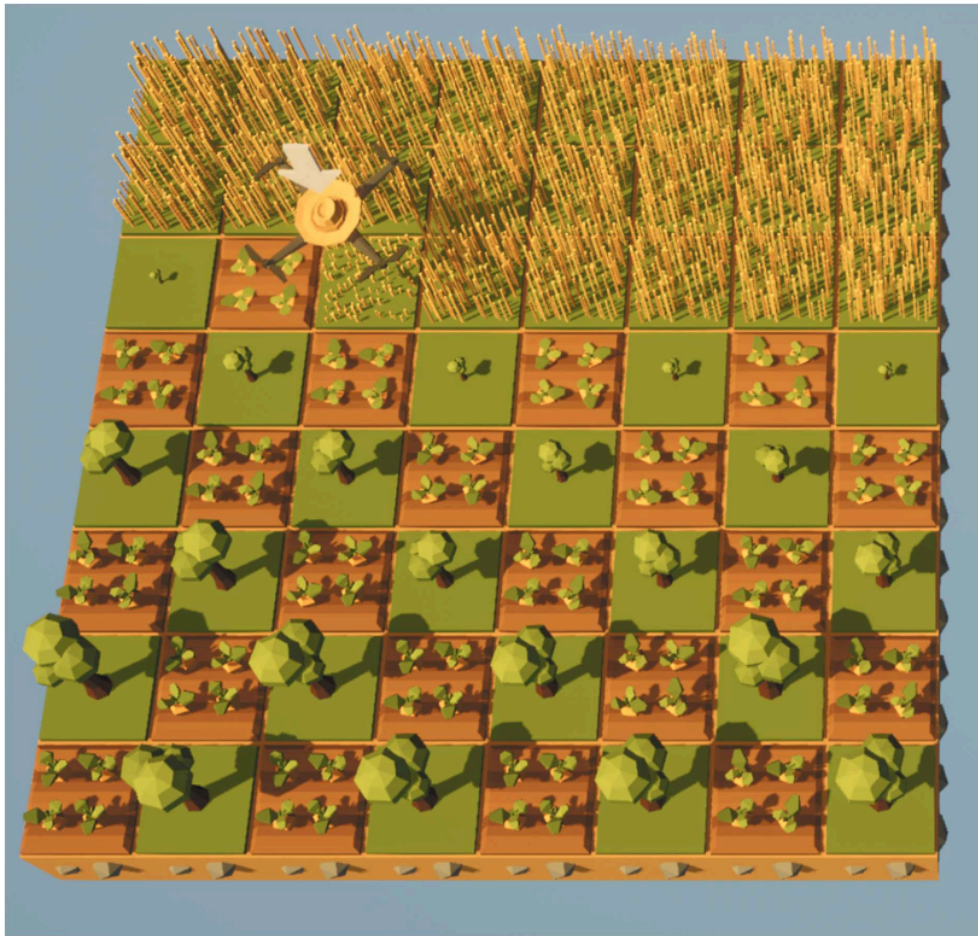


Figure 17 - The Farmer Was Replaced

The situation within the video game required a solution to effectively “farming” carrots and trees; distributing them in certain ways on the grid improves “harvesting” depending on the users’ preferences. This situation demonstrates the use of the logical “or” operation with its Python inspired syntax, resulting in an increased understanding and competency in which the flowchart shown in Figure 1, states (if occupancy_status == 0 or occupancy_status == 1).

In conclusion to the test cases, several opportunities to improve and optimise the original code were identified. Test Case B, as shown in Figures 4 and 13, presented the most negatively impactful issues regarding the original code logic. The results shown in the Test Case tables represent that (max_free_spaces) was incorrect for

the purpose of the program; resulting in changing the variable to 0 as opposed to 1. This became clear upon performing the provided test cases.

Test Case D presented a particular representation of, although the mathematical and logical result is correct, for users of the program, it may be sub optimal in terms of the most desirable result.

In contrast to Test Case A, Figures 3 and 12, this Test Case scenario resulted in the most accurate output. Only minor user communication features were included in the final code in Figure 12, such as “free spaces” for the print function in Python.

Further investigations could be recommended regarding other smart parking programming scenarios. Test Cases C as shown in Figures 5 and 14 provides an opportunity for the program to become more appropriate for professional or industry level use, as opposed to solely academic. [1]

7. References

- [1] JetBrains, "PyCharm: The only Python IDE you need," JetBrains, 2026. [Online]. Available: <https://www.jetbrains.com/pycharm/features/>. [Accessed 26 January 2026].
- [2] I. Putten, M.V. Putten, Python Illustrated, Birmingham: Packt Publishing Ltd., 2026.
- [3] M. McGrath, Python in easy steps : makes programming fun!, 2nd ed., In Easy Steps Limited, 2018, pp. 15-16.
- [4] D. Novikov, Python for Beginners, n.p., 2019, pp. 39-42.
- [5] Refsnes Data AS, "Python String Formatting," 2026. [Online]. Available: https://www.w3schools.com/python/python_string_formatting.asp. [Accessed 30 April 2026].
- [6] Tutorial Reference, "How to Resolve Python Error "ValueError: invalid literal for int() with base 10"," 2026. [Online]. Available: <https://tutorialreference.com/python/examples/faq/python-error-valueerror-invalid-literal-for-int-with-base-10>. [Accessed 12 May 2026].
- [7] Python, "What is Python? Executive Summary," Python, 2026. [Online]. Available: <https://www.python.org/doc/essays/blurbl/>. [Accessed 5 May 2026].
- [8] M. McGrath, C Programming: grasp the nuts and bolts of programming, 5tg ed., In Easy Steps, 2018, p. 58.
- [9] T. Herzog, "The Farmer Was Replaced," Metaroot, Zürich, 2025.

Laurel Black Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

80/100

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

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

GRADING FORM: ACCA4020 RUBRIC 2026

LAUREL BLACK

80

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]



Most of the flowchart logic is implemented and test cases are shown with minor gaps. The error related to Spaces_per_Zone has not been reported.

15

TASK 1B [15]

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



Some errors were identified, but important issues, such as equal-zone logic, are not explained.

11

TASK 2A [20]

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



Most errors are fixed, including validation and general logic improvements. However, handling of the equal zone scenario or overall logic could be more robust

16

TASK 2B [20]

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



Most test cases are re-tested with generally correct outputs. Improvements are evident, though comparisons or explanations could be more detailed

16

TASK 3 [10]

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



Good explanation with some insight into design challenges.

8

TASK 4 [15]

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



A clear reflection covering key learning points from the assignment. Demonstrates understanding of errors and fixes

14