

Introduction to computer programming

Jayce atkins 2401799

ACHA4020

Introduction

This assignment's focus is on designing, testing and improving a smart car park system for a university. They have a multizone car park where the system records which car parking spaces are occupied and which are free. It also calculates the capacity level of each zone, displays warnings when zones are over 80% capacity and shows which zones are the least occupied.

Universities have to normally manage large amounts of vehicles across multiple parking zones. Manual monitoring is extremely inefficient with large amounts and this can lead to inaccurate free space tracking numbers. Automated parking management improves organisation and allows the staff to quickly identify available spaces.

The first part of the assignment given uses the provided flowchart exactly as given even though it contains logic errors. The program was tested then using the given test cases to identify the errors. After this a corrected and amended version was tested using the same test data again to make sure all issues were resolved.

Programming logic and flowcharts are a key skill in programming and university, they are also useful in careers later on. They allow developers to plan systems before implementation begins within a project,

The second part of the assignment is a reflection on the assignment including how skills were improved and how my previous knowledge was expanded and solidified followed by the conclusion.

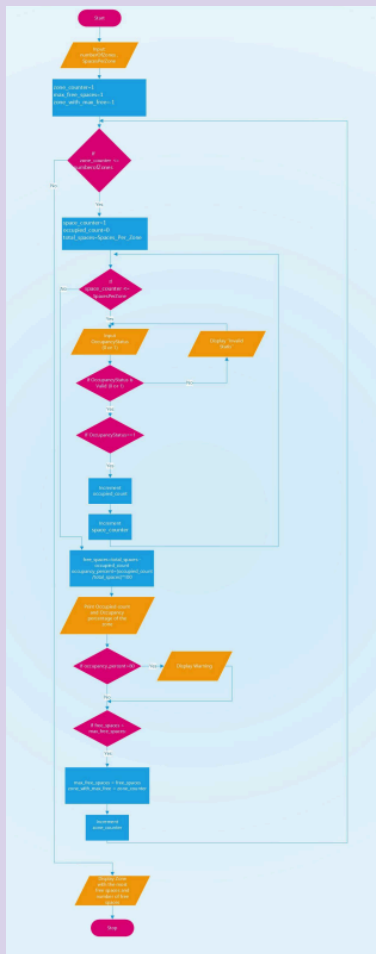
Task 1

Check screenshots before turn in may have made an error need pc to fix (may have made file mixup)

This first task was to implement the flowchart (below) as code (done in python) exactly as given without correcting errors. At this point the purpose isn't to improve or correct the design but to change the flow chart into working python code and see how the programme behaves in testing

The program is made to simulate an automated parking management system for a uni car park that has multiple car parking zones. Each parking zone has a fixed number of spaces available in a car park. This system asks the user to enter whether each parking space is free by entering 1 or 0

Multiple variables were needed to control the program and store calculations during running. The variable `NUMBER_OF_ZONES` stores the total amount of parking zones entered by the user.



```

assignment 2 - broken code.py
1 number_of_zones = int(input("Enter number of zones: "))
2 spaces_per_zone = int(input("Enter number of parking spaces per zone: "))
3 zone_counter = 1
4 max_free_spaces = 0
5 zone_with_max_free = 1
6 while zone_counter <= number_of_zones:
7     space_counter = 1
8     occupied_count = 0
9     total_spaces = spaces_per_zone
10    print("\nZone", zone_counter)
11    while space_counter <= spaces_per_zone:
12        occupancy_status = input("Enter occupancy status for space " + str(space_counter) + " (1 = occupied, 0 = free): ")
13        if occupancy_status == "0" or occupancy_status == "1":
14            if occupancy_status == "1":
15                occupied_count = occupied_count + 1
16                space_counter = space_counter + 1
17            else:
18                print("Invalid data")
19                space_counter = space_counter + 1
20        free_spaces = total_spaces - occupied_count
21        occupancy_percent = (occupied_count / total_spaces) * 100
22        print("Occupied spaces:", occupied_count)
23        print("Occupancy percentage:", occupancy_percent, "%")
24        if occupancy_percent > 80:
25            print("Warning: occupancy exceeds 80%")
26        if free_spaces > max_free_spaces:
27            max_free_spaces = free_spaces
28            zone_with_max_free = zone_counter
29        zone_counter = zone_counter + 1
30    print("\nZone with the most free spaces:", zone_with_max_free)
31    print("Number of free spaces:", max_free_spaces)
32

```

This code was then tested using the four given test cases, these cases helped to point out the errors and bugs in the code that were fixed in task 2.

Test case A

Zones - 2

Spaces - 4

Zone 1 - 1, 1, 0, 0

Zone 2 - 0, 0, 1, 1

Input display

```
Enter number of zones: 2
Enter number of parking spaces per zone: 4

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 1
Enter occupancy status for space 3 (1 = occupied, 0 = free): 0
Enter occupancy status for space 4 (1 = occupied, 0 = free): 0
Occupied spaces: 2
Occupancy percentage: 50.0 %

Zone 2
Enter occupancy status for space 1 (1 = occupied, 0 = free): 0
Enter occupancy status for space 2 (1 = occupied, 0 = free): 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Enter occupancy status for space 4 (1 = occupied, 0 = free): 1
Occupied spaces: 2
Occupancy percentage: 50.0 %
```

Expected behaviour for this version was

Zone 1 occupied spaces : 2

Zone 1 occupancy percentage : 50%

Zone 2 occupied spaces - 2

Zone 2 occupancy percentage - 50%

Zone with most free spaces : zone 1

Number of free spaces : 2

Both zones have the same number of free spaces however the program keeps zone 1 as it only updates when a zone has more free spaces and not equal free spaces.

Results

Both zones have the same number of free spaces however the program keeps zone 1 because it only updates when another zone has more free spaces and not equal.

```
Zone with the most free spaces: 1
Number of free spaces: 2

Process finished with exit code 0
```

Test case B

Zones - 2

Spaces - 3

Zone 1 - 1, 1, 1

Zone 2 - 1, 1, 1

When this was run the results predicted were

Zone 1 occupied spaces : 3

Zone 1 occupancy percentage - 100%

Warning - occupied spaces over 80%

Zone 2 occupied spaces - 3

Warning - Zone 2 occupancy is over 80%

Zone with most spaces free - Zone 1

Number of free spaces - 0

There may be an issue here as both zones have no free spaces, but the programme still displays zone 1 as a default.

Input display & results

```
Enter number of zones: 2
Enter number of parking spaces per zone: 3

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 1
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 3
Occupancy percentage: 100.0 %
Warning: occupancy exceeds 80%

Zone 2
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 1
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 3
Occupancy percentage: 100.0 %
Warning: occupancy exceeds 80%

Zone with the most free spaces: 1
Number of free spaces: 0

Process finished with exit code 0
```

There is a small logic error as predicted here as both zones each have 0 free spaces but the program still reports zone 1 as the one with the most free spaces as zone_with_max_free starts at 1 and does not change when free spaces are equal with each other.

Test case C -

Zones - 1

Spaces - 3

Occupied spaces - 1, 2, 0, %

Predicted results here were as shown below

1 accepted

2 invalid data

0 accepted

% invalid data error

Occupied spaces - 1

Occupancy percentage 25%

Zone with most free spaces : 3

Input display and results

```
Enter number of zones: 1
Enter number of parking spaces per zone: 4

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 2
Invalid data
Enter occupancy status for space 3 (1 = occupied, 0 = free): 0
Enter occupancy status for space 4 (1 = occupied, 0 = free): %
Invalid data
Occupied spaces: 1
Occupancy percentage: 25.0 %

Zone with the most free spaces: 1
Number of free spaces: 3
```

The inputs "2" and "%" are rejected however the program still continues to the next parking space. This means the program is giving a count of spaces incorrectly as invalid data is still counting towards the total number of spaces.

Test case D

Zones - 2

Spaces - 3

Zone 1 spaces - 1, 0, 1

Zone 2 - 0, 0, 1

This test is predicted to work correctly, with the behaviour being

Zone 1 occupied spaces : 2

Zone one occupancy % - 66.67%

Zone2 taken spaces - 1

Zone 2 occupied percentage - 33.33%

Zone with most free space - 2
 Number of free spaces available : 2

Input display and results

```

Enter number of zones: 2
Enter number of parking spaces per zone: 3

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 2
Occupancy percentage: 66.66666666666666 %

Zone 2
Enter occupancy status for space 1 (1 = occupied, 0 = free): 0
Enter occupancy status for space 2 (1 = occupied, 0 = free): 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 1
Occupancy percentage: 33.33333333333333 %

Zone with the most free spaces: 2
Number of free spaces: 2

Process finished with exit code 0

```

This data test works correctly with zone 2 being displayed as the most free spaces as the program updated with `zone_with_max_free` correctly.

Error analysis table

Test case	Observed output	Error type	Explanation
A	Both zones were at 50% capacity but zone 1 was displayed as having the most free spaces	Minor logic issue	Both zones has the same number of free spaces, but the program only updates the zone when a bigger value is found.

			Due to this zone 1 remains selected
B	Both zones showed at 100% occupancy and displayed warning messages. Zone 1 was presented as having the most free car park spaces	Logic issue	Both zones had 0 free spaces but the program still displayed zone 1 because zone_with_max_free starts at 1 and doesn't change when values are equal.
C	Invalid inputs "2" and "%" displayed as invalid data but the program still continued to the input for the next space.	Validation error	Invalid entries should be rejected and reentered. However the program still increases the counter after invalid inputs making false calculations
D	Free spaces and percentages were calculated correctly. Zone 2 was correctly shown as having the most free spaces	No error	This test case worked as intended because all inputs were valid and one zone had more free spaces than the other.

Table 1

This testing processes showed several logical and validation issues in the original flowchart design. The most significant problem occurred during test C with invalid data. Where the programme accepted incorrect inputs and continued processing instead of requesting the correct data value. Some minor other logic issues were also identified when zones contained the same free number of car parking spaces.

Task 2 - corrected program

Fix 1 - when the user enters invalid data, the program shouldn't move onto the next parking space. Instead it should ask for the same space again until a valid data is entered.

```

assignment 2 - broken code.py  assignment 2 - corrected code.py ×
1 number_of_zones = int(input("Enter number of zones: "))
2 spaces_per_zone = int(input("Enter number of parking spaces per zone: "))
3 zone_counter = 1
4 max_free_spaces = -1
5 zone_with_max_free = 1
6 while zone_counter <= number_of_zones:
7     space_counter = 1
8     occupied_count = 0
9     total_spaces = spaces_per_zone
10    print("\nZone", zone_counter)
11    while space_counter <= spaces_per_zone:
12        occupancy_status = input("Enter occupancy status for space " + str(space_counter) + " (1 = occupied, 0 = free): ")
13        if occupancy_status == "0" or occupancy_status == "1":
14            if occupancy_status == "1":
15                occupied_count = occupied_count + 1
16                space_counter = space_counter + 1
17            else:
18                print("Invalid data, please enter 1 or 0")
19        free_spaces = total_spaces - occupied_count
20        occupancy_percent = (occupied_count / total_spaces) * 100
21        print("Occupied spaces:", occupied_count)
22        print("Free spaces:", free_spaces)
23        print("Occupancy percentage:", occupancy_percent, "%")
24        if occupancy_percent > 80:
25            print("Warning: occupancy exceeds 80%")
26        if free_spaces > max_free_spaces:
27            max_free_spaces = free_spaces
28            zone_with_max_free = zone_counter
29        zone_counter = zone_counter + 1
30    print("\nZone with the most free spaces:", zone_with_max_free)
31    print("Number of free spaces:", max_free_spaces)

```

Corrected version test case A

Input display and results

```
"D:\pythonProjects\codebasics assignment 2\.venv\Scripts\python.exe" "D:\pythonProjects\codeb
Enter number of zones: 2
Enter number of parking spaces per zone: 4

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 1
Enter occupancy status for space 3 (1 = occupied, 0 = free): 0
Enter occupancy status for space 4 (1 = occupied, 0 = free): 0
Occupied spaces: 2
Free spaces: 2
Occupancy percentage: 50.0 %

Zone 2
Enter occupancy status for space 1 (1 = occupied, 0 = free): 0
Enter occupancy status for space 2 (1 = occupied, 0 = free): 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Enter occupancy status for space 4 (1 = occupied, 0 = free): 1
Occupied spaces: 2
Free spaces: 2
Occupancy percentage: 50.0 %

Zone with the most free spaces: 1
Number of free spaces: 2

Process finished with exit code 0
```

Test case B

Input display and results

```
"D:\pythonProjects\codebasics assignment 2\.venv\Scripts\python.exe" "D:\p
Enter number of zones: 2
Enter number of parking spaces per zone: 3

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 1
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 3
Free spaces: 0
Occupancy percentage: 100.0 %
Warning: occupancy exceeds 80%

Zone 2
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 1
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 3
Free spaces: 0
Occupancy percentage: 100.0 %
Warning: occupancy exceeds 80%

Zone with the most free spaces: 1
Number of free spaces: 0

Process finished with exit code 0
```

Test case C

Input display and results

This time the program asks for the spaces again in the correct format

```
"D:\pythonProjects\codebasics assignment 2\.venv\Scripts\python.exe" "D:\pythonF
Enter number of zones: 1
Enter number of parking spaces per zone: 4

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 2
Invalid data, please enter 1 or 0
Enter occupancy status for space 2 (1 = occupied, 0 = free): 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): %
Invalid data, please enter 1 or 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Enter occupancy status for space 4 (1 = occupied, 0 = free): 1
Occupied spaces: 3
Free spaces: 1
Occupancy percentage: 75.0 %

Zone with the most free spaces: 1
Number of free spaces: 1

Process finished with exit code 0
```

Test case D

Input display & results

```
"D:\pythonProjects\codebasics assignment 2\.venv\Scripts\python.exe" "D:\pythonProje
Enter number of zones: 2
Enter number of parking spaces per zone: 3

Zone 1
Enter occupancy status for space 1 (1 = occupied, 0 = free): 1
Enter occupancy status for space 2 (1 = occupied, 0 = free): 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 2
Free spaces: 1
Occupancy percentage: 66.66666666666666 %

Zone 2
Enter occupancy status for space 1 (1 = occupied, 0 = free): 0
Enter occupancy status for space 2 (1 = occupied, 0 = free): 0
Enter occupancy status for space 3 (1 = occupied, 0 = free): 1
Occupied spaces: 1
Free spaces: 2
Occupancy percentage: 33.33333333333333 %

Zone with the most free spaces: 2
Number of free spaces: 2

Process finished with exit code 0
```

Corrected program results table

Test case	Output seen	Yes issue resolved?	Explanation
a	Both zones showed 2 occupied spaces, 2 free spaces and 50% occupied. Sonze 1 was displayed as the most free spaces	yes	The programm correctly calculated the taken and free spaces. Both zones had equal free spaces so still zone 1 was remained selected for display
b	Both sonzes showed 3 taken spaces and so were 100% occupied. A warning was showed for both zones	yes	The program correctly identified that both zones were full and displayed the warning because the taken spaces

			were over 80%
C	Invalid inputs were rejected and the user was asked to enter the value again. The final result showed 3 taken spaces and 1 free space for 75% occupancy	yes	The validation error was fixed because invalid inputs no longer increased the spaces counted and the program asks for correctly format data to be in
D	Zone one showed two occupied and 1 free space. Zone 2 showed 1 occupied and 2 free spaces. Zone two was shown as having the most free spaces.	yes	The program identified zone 2 as having the most free spaces and displayed that.

Table 2

After correcting the program all test cases produced suitable results. The main issues in the original version were fixed by preventing invalid inputs from increasing the free spaces counter. This means the program now asks the user to reenter invalid data until a correct value is provided, the corrected version also calculates occupied spaces, free spaces and occupied percentages correctly.

Task 3

The logic of the program can be improved by adding lists to store the results for each zone. In the program at the moment separate variables such like *occupied_count*, *free_spaces* and *zone_with_max_free* are used while the program is running.

A list would allow the program to store the number of free spaces for each zone. This would make it easier to compare zone capacity, checking for equal results and displaying a summary at the end. The validation logic could also be improved by using a loop that only accepts "0" or 1. Making the program more reliable as it won't accept anything other than the correct data format.

Overall using clearer variable names, lists and stronger validation would make the program easier to understand improve and test in future.

Sample

```
Free_spaces_list = []
free_spaces_list.append(free_spaces)
```

Adding this within the code ,, this would store each zones number of free spaces in a list

Task 4 - learning reflections

These activity's and tasks helped to improve my base coding and python language skills and understanding in programming and problem solving. I reaffirmed and added to my previous knowledge of using loops to repeat tasks and how nested loops can be used to process multiple things at once, in this case multiple parking zones and spaces for a uni versity car park.

One of the most important notes was the importance of validation. During the first version of the program, invalid inputs caused incorrect results as the counter continued even with wrong data inputted. After correcting the program i better understood how validation improves the accuracy and reliability of the system.

The testing process also helped me to better my error spotting and understand how programmers find and fix logical errors smoothly. By using different test cases i was able to compare the expected results vs the actual and improve the program step by step.

Overall this assignment improved my confidence and skills in python programming and helped me to better understand how flowcharts can be converted into better working and functional code.

Conclusion

In conclusion this assignment involved creating, testing and improving a smart parking system for a uni car park. The original flowchart given in the material was first done as code exactly as provided and then testewd using four test cases. This process helped to show logical and validation problems within the code.

A corrected version of the program was developed to solve these issues. The updated program successfully validated input formatss, calculated capacity level correctly, displayed warnings when the capacity reacher 80% or higher and correctly identified the most open zone.

When testing and debugging the assignment demonstrated the importance of validation, logical thinking when coding and structured programming when debveloping reliable software systems.