Debugging & Tracing (8 points)

Consider the following (buggy) code:

```python
def divide_and_round(n):
    """
    Divides an integer n by 2 and rounds up to the nearest whole number
    """
    if n % 2 == 0:
        n = n / 2
    else:
        n = (n + 1) / 2

def main():
    n = 42
    divide_and_round(n)
    print(n)  # should print 21
```

Part A (4 points):

What will be printed at the end of the program? Why?

Part B (4 points):

How can this program be fixed to print 21? You can make changes to both `divide_and_round` and `main`, but `divide_and_round` must still implement the functionality described in its comment for all possible values of its parameter. You should write a fixed version of the program with comments indicating each line you changed.
Odd numbers (6 points)
Write a program that prints the first 100 odd numbers greater than 0. In other words print out 1 then 3, then 5 all the way up to, and including, 199.

Can I Ride the Rollercoaster? (6 points)
Write a program which asks the user for their height in meters (which is a real number).
If their height is less than 1 meter or greater than 2 meters, print "You can't ride the roller coaster".
If their height is between 1 meter and 2 meters print "You can ride the roller coaster".
Here are a few examples. User input is in blue

Enter height in meters: 1.2
You can ride the roller coaster.

Enter height in meters: 2.5
You can't ride the roller coaster.
Ramp Climbing Karel (15 points)

Write a program that has Karel draw a diagonal line across the world, with a slope of 1/2, like so:

A few notes
- Karel always begins at the bottom left corner of the world facing East.
- Karel's bag has infinite beepers.
- It does not matter which direction Karel ends up facing.
You may assume that the world is an **odd number** of columns across.

We've provided you implementations of the `turn_right` and `turn_around` functions, although you aren't required to use either of them.

You **must not** use any non-Karel features like variables (other than `i` in a `for` loop), `return` or `break`. You may use any Karel features described in the course reader.
A simple way to achieve great things in life is to make small forward progress every day. Non-decreasing progress is one of the principles behind modern AI.

Write a program that asks the user to enter a sequence of "non-decreasing" numbers one at a time. Numbers are non-decreasing if each number is greater than or equal to the last.

When the user enters a number which is smaller than their previously entered value, the program is over. Tell the user how long their sequence was.

Here is an example (values entered by the user in blue):

```
Enter a sequence of non-decreasing numbers.
Enter num: -1
Enter num: 0
Enter num: 1
Enter num: 3.12
Enter num: 99
Enter num: 99
Enter num: 42
Thanks for playing!
Sequence length: 6
```
A few notes:

- Include the intro message, Enter a sequence of non-decreasing numbers.
- Include the exit message, Thanks for playing!
- The length of the sequence does not include the very last number entered as it is "decreasing"
- Your program should accept floating point numbers.
- The sequence only ends when one number is strictly less than the last.
- You do not have to handle the case where the user enters a value which is not a number (say "abc").
- The user can start with any number, positive or negative.
- Note that the shortest possible sequence is 1. A sequence with a single number can't be decreasing. A sequence with two numbers can be decreasing.

More examples:

Enter a sequence of non-decreasing numbers.
Enter num: 5
Enter num: 4
Thanks for playing!
Sequence length: 1

Enter a sequence of non-decreasing numbers.
Enter num: 1
Enter num: 2
Enter num: 3
Enter num: 4
Enter num: 2
Thanks for playing!
Sequence length: 4

Enter a sequence of non-decreasing numbers.
Enter num: -5
Enter num: -5
Enter num: -5
Enter num: -6
Thanks for playing!
Sequence length: 3