Assessment Task 1: Client's Briefing #2 Partial Transcript

Hello, it's Elton, your *QUT Data Services* representative, again. I wanted to thank you for submitting your symbol design on time. I've sent it to our back-room boffins for evaluation and they will get their verdict back to you as quickly as possible. (Unfortunately, however, we do have some other programmers who are submitting their own designs late, for various reasons, and that will delay the overall evaluation process.)

However, given this show of faith, I'm now authorised to entrust you with the data sets we need visualised. Time is of the essence because we need your complete, finished program by the end of Week 7.

Our customers are data analysts who need to understand logs that record the way certain objects attempt to move around in a monitored area. These logs appear as lists of numbers and text strings, which makes it difficult to understand exactly what the object does. Your job, therefore, is to write a function which displays the logs' content visually, in an easily interpreted form. When you run our Python template you'll remember that some cells are marked by dots.

[*Runs the template file from the last briefing and points out the special cells*]

The middle dot, in cell E5, is the exact centre of the grid and that's where the object always starts, pointing in one of the six possible directions.

From there it moves one cell at a time but stops when any one of three things happens. It may run out of energy. It may move outside of the grid, beyond our surveillance perimeter. Or it may encounter one of the three special cells, A3, D8 or H6.

So what we need now are some data sets which tell us which direction the object moves in each step. To provide you with data sets to visualise, our data analysts have created a Python module called `assignment_1_data_source`, which you can find accompanying this briefing. You're welcome to study this code but do not change any of it.

[*Shows the contents of the* `assignment_1_data_source` *module*]

If you put this module into the same folder as the original Python template we gave you, it will cause the program to generate a new data set each time the program is run. Let's try it.

[*Runs the template file a few times with the data module in place*]

As you can see, each time you run the program you get a different data set. These are the data sets of interest to our analysts, but in this form they're very hard to understand. It's not immediately obvious by looking at them where the object goes.

Your program is going to solve this problem for us, by visualising the data sets using the hexagonal grid.

Let's have a look at a typical data set.

[*Points out the features of a particular data set*]

In summary, each data set generated consists of a sequence of moves. The first value in each move is a counter that begins at zero and increases by one with each move generated.

Move number zero is different from the others because it tells us about the object's initial state. It has a number which tells us how much energy the object has at the beginning. This is how many moves the object can make before it becomes exhausted. The next value is the object's initial orientation, which can be any of the six directions, north, north-east, south-east, etc.

The rest of the moves, after the first one, all have the same format and consist of the move number and the action the object tries to perform in that move. There are three possible actions.

- "Move forward" means the object moves one grid cell in whichever direction it's facing.
- "Move & turn right" means the object moves one grid cell in whichever direction it's facing and turns 60 degrees to the right.
- "Move & turn left" means the object moves one grid cell in whichever direction it's facing and turns 60 degrees to the left.

Your program must draw the object making each of the moves in the data set until one of three things happens.

1. One possibility is that the object runs out of energy and can't do any more moves.
2. Another possibility is that the object's move would take it outside of the grid, where we can't see it, i.e., it's gone beyond the area under observation.
3. And the third possibility is that the object moves onto one of the three special cells, apart from the home cell (A3, D8 and H6).

In all three of these cases the simulation stops. Importantly, though, it won't be immediately obvious to the data analyst *why* the object stopped moving, so you must write a clear message at the top of the drawing canvas explaining what caused it to stop.

However, what each of these three "stopping conditions" means depends on which kind of object you have chosen to draw. This is why it was so important that you chose an appropriate object, capable of moving under its own power, in any one of the six directions.

To show you what's expected our back-room boffins have extended their sample solution, so I'll run their demonstration on a few different data sets. Remember that their object was the ladybird from the nursery rhyme, *Ladybird, ladybird, fly away home*, so they have interpreted the three stopping conditions in that context.

Let's look at some examples where the object runs out of energy.

[*Shows an example where the object runs out of energy after three moves* (Seed 37056)]

[*Shows an example where the object runs out of energy on the edge of the grid so doesn't leave it* (Seed 43879)]

[*Shows an example where the object has no energy at the beginning of the simulation so doesn't move at all* (Seed 89704)]

Another way in which the simulation can end is if the object moves outside of the grid. Let's have a look at a few examples of that.

[*Shows an example where the object leaves the grid via the bottom in the third move* (Seed 71185)]

[*Shows an example where the object leaves the grid via the left border in the fifth move* (Seed 6449)]

Those are two ways in which the simulation can end but there's a third. The other possibility is that the object encounters one of the three special cells, A3, D8 or H6, and the simulation should stop as soon as that heppens. Let's have a look at some examples.

[*Shows an example where the object encounters special cell H6* (Seed 82830)]

[*Shows an example where the object encounters special cell D8 and simultaneously runs out of energy* (Seed 53162)]

Most of the data sets produced are quite short, but with a bit of perseverence, running the program a few times, you can often find a data set containing a longer sequence of moves. Let's have a look.

[*Shows an example where the object completes eight move after the initialisation step* (Seed 26907)]

In summary, therefore, you need to follow the moves in the data sets in the same way as these examples, but using the symbol you designed for us earlier, of course.

- Your code must start the object in the centre cell, oriented as per the instruction in Move 0.
- After that it must draw the object in its new location and orientation as instructed in each move until the object
    - runs out of energy,
    - leaves the area under observation, or
    - encounters one of the three special cells.
- And finally, you need to write a message above the grid, telling the viewer why the object stopped moving.

Of course, the messages that you write depend on your particular chosen theme. *You* have to decide what it means if the object reaches one of the special cells or leaves the grid or becomes exhausted. So explain why the simulation stopped in the context of the particular object you've chosen to draw.

To help you, each data set generated is printed to the shell window, as you have just seen. More importantly, however, the data set is also returned as a list by function `data_set`, which makes it available to your `visualise_data` function. Let's have a look at the code.

[*Shows the call to* `data_set` *in the template's main program*]

Your task is to complete the `visualise_data` function; *all* of your code goes in or is called by this function, where the **`pass`** statement appears.

You also need to ensure that your code does *not* call function `data_set` because it's *already* called in the main program. The function call to `data_set` in the main program

- produces the random data sets,
- displays them in the shell window, and
- gives them to your `visualise_data` function.

Finally, we fully appreciate how difficult it is to work with large data sets that change randomly. To help you develop your program with some unchanging data sets, our boffins have therefore incorporated a feature into the Python template code that allows the data sets to be fixed temporarily. If you add an integer argument in the call to function `data_set` in the main program it will always generate the same list of values.

For instance, if you put seed value 78655 into the call to `data_set` it will always produce a list of moves in which the object runs out of energy in the bottom-left corner of the grid. Let's see how it works with the boffins' sample solution.

[*Shows how to use* Seed 78655 *to produce a repeatable behaviour*]

In the new data source module we've listed many such seeds showing what behaviour to expect when you use that seed as the argument for function `data_set`.

[*Shows the "seed" values documented in the data source module*]

You can use this information to help debug your program because it tells you exactly what the object should do for each of the seeds.

However, when you submit your final program you must not use a fixed value for the data sets. Your program must work for *any* possible data set that function `data_set` can produce. When you submit your code, make sure there is no argument in the call to `data_set` at the bottom of the template file.

Another helpful feature that the boffins have introduced is to display the seed value in the shell window every time you run your program, so that you can replay any behaviour of interest simply by copying that value and putting it in the call to `data_set` while you're developing your code.

[*Removes the seed value in the call to* `data_set` *and runs the program several times to get different behaviours*]

When you've completed your program, please upload it to our organisation's IT system just as you did before. We only need the single Python file, `assignment_1.py`. Don't send us a copy of the configuration or data source modules because obviously we already have them, and we'll use our own copies to test your solution.

As usual, to ensure your solution is portable across any computing platform, it must be written in standard Python 3, and must not rely on any extension modules that need to be downloaded and installed separately.

Also, we welcome receiving, and encourage you to submit, multiple early drafts as you develop your solution, as insurance against technical failures, floods, pandemics, and so on that could occur before the end of Week 7. If you can't complete the whole job in time we'll give you partial credit for the parts you do complete.

That's all for this set of requirements. Our company is looking forward to receiving your updated program at your earliest convenience. Goodbye.