Snowflake Generation

Valerie Maxville[1]

Curtin University, Perth, WA, Australia
v.maxville@curtin.edu.au
http://www.curtin.edu.au/

Abstract. For over fifty years we have worked to improve the teaching of computer science and coding. Teaching computational science extends on these challenges as students may be less inclined towards coding given they have chosen a different discipline which may have only recently become computational. Introductory coding education could be considered a checklist of skills, however that does not prepare students for tackling innovative projects. To apply coding to a domain, students need to take their skills and venture into the unknown, persevering through various levels of errors and misunderstanding. In this paper we reflect on programming assignments in Curtin Computing's Fundamentals of Programming unit. In the recent Summer School, students were challenged to simulate the generation and movement of snowflakes, experiencing frustration and elation as they achieved varying levels of success in the assignment. Although these assignments are resource-intensive in design, student effort and assessment, we see them as the most effective way to prepare students for future computational science projects.

Keywords: Education · Computational Science · Programming · STEM.

1 Introduction

There has been a movement in recent years pushing for everyone to learn how to code. A recent post by the Forbes Technology Council [1] cited twelve of fifteen members supporting coding for everyone. It's a great idea, however, teaching coding isn't always easy. Huggard [2] writes of negative attitudes and "programming trauma" as barriers to learning. Initiatives such as CoderDojo seek to raise enthusiasm through coding clubs [3],[4], aiming to take students through the levels of Digital Proficiency (Figure 1). These clubs are aimed at school children to create and/or extend an interest in coding, aiming prepare students for future employment opportunities. There is a question on the long-term impact of programming and STEM interventions, and further study is required to gauge what duration and regularity is required for students to maintain interest [5].

In the past, university level coding was primarily confined to Computer Science and Information Technology courses. These introductory programming classes have a long history of low pass-rates along with the challenge of assessing programming "ability" [6]. At Curtin University, we are seeing rapid growth in the number of students taking programming courses, whether through choosing

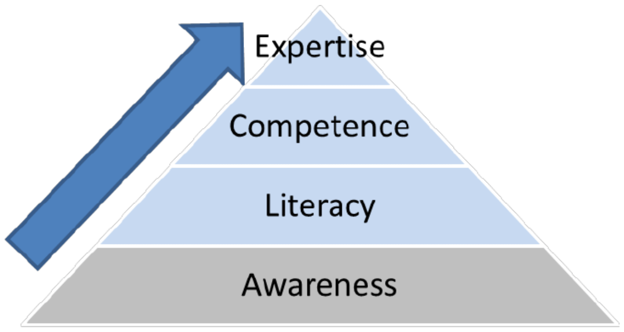


Fig. 1: Digital Proficiency Levels (adapted from ECOL Foundation, 2011) [4]

computing degrees; finding their courses now include programming; or taking the units as electives. Engineering students need at least one full unit of programming for their degree to be accredited, and science students are also required to take programming in their first year. So, how do we adapt our courses to be suitable for a wider range of backgrounds, and even more diverse fields of application? There have been initiatives in the computational science area, including [7] and [8], which highlight key skills and concepts required for science students and researchers.

This increase in students does not occur in a vacuum, and we have additional pressures to accommodate. Directives to reduce expenses can have a detrimental impact on staff : student ratios at exactly the time that we find ourselves struggling to scale teaching. In addition, a mandate to reduce the number of assessments forces changes that are not based on improving outcomes or learning, and may indeed have the opposite effect. The media will have us believe that the students are part of a "snowflake generation", unable to cope with the stresses of life - which would include study. However, that stigmatises the challenges many are having with mental health in an unpredictable world [9]. This is compounded by the challenges of a worldwide pandemic, affecting the health, families and livelihoods of students. And those teaching through these time, requiring agility to move quickly between teaching modes and the loss of the 1-1, face to face interaction that often makes all the difference.

In this paper, we discuss approaching these challenges through a snowflake-related assessment, part of an introductory programming unit taken by science and engineering students. As the students persevered through the assignment and the unit, each of their journeys was unique, forming a delicate network of connections of ideas: snowflakes, in a good way.

"Snowflakes fascinate me... Millions of them falling gently to the ground...
And they say that no two of them are alike! Each one completely different

from all the others... The last of the rugged individualists!" – Charles M. Schulz [1]

2 Fundamentals of Programming

The unit, Fundamentals of Programming, was developed in 2017 for new courses in Predictive Analytics and Data Science at Curtin University. The predicted number of students in its first run was 20. By the time semester started there were 120 students enrolled. Since becoming part of the first year core for both science and engineering course, the unit is now attracting over 700 students at the main campus, and is delivered in Malaysia, Mauritius, Dubai and Sri Lanka.

As a new course, that author was given Unit Learning Objectives as a guide to what was required in the unit. The language was to be Python were clearly looking for a "vanilla" introductory programming unit. After eight years as the Education Program Leader at the Pawsey Supercomputing Centre, the author had observed the skills and tools used by a range of computational scientists, and the key issue in data science raised by leaders in eResearch and digital humanities. The science students would need to learn about scripting and automation; notebooks and reproducibility; software carpentry; and would need to work in a Linux environment.

In terms of assessment, a typical 50% exam, 20% assignment, 15% mid-semester test and 15% practical test was in place. Students needed to submit a reasonable attempt for the assignment, and receive 40% on the exam to pass the unit. From the start, the "practical test" was reworked to be assessed as a series of 5 practical tests worth 3% each. The tests were to be competency-based, with students allowed to resubmit until they could demonstrate the required skills. Student response to this approach was overwhelmingly positive. The assignment required the demonstration of the work, aiding Academic Integrity.

We were then given a directive from the university that no unit could have more than three assessments. An initial exemption was approved, but eventually the mid-semester test was dropped and the assignment and practical test weightings increased to 30% and 20% respectively. With the pandemic, the exam could no longer be held in a large, controlled examination venue. In response, the exam was replaced with a Final Assessment: 24-hour open book, practical and delivered in Jupyter Notebook. In the current offering of the unit, the assignment weighting has been increased and the Final Assessment decreased to both be 40% to better reflect the effort required. This is in response to students questioning the weighting with respect to the time required.

The unit is both easy and hard. It is hard in that you cannot pass without doing the work. It is easy, in that it is very scaffolded, and if you do the work, you will pass. With Python, plotting is trivially easy, and is used from week three to provide a visual representation of coding outcomes - a huge advantage

[1] Snowflake quotes sourced from: `https://kidadl.com/articles/best-snowflake-quotes-that-are-truly-unique` and `https://routinelynomadic.com/snowflake-quotes-sayings-captions/`

compared to Java and other languages where graphics sits beyond a barricade of pre-requisite knowledge. We have quite a few experienced coders taking the unit, but even more of the students have no coding experience. Given this range of backgrounds, we work hard to keep the unit challenging so that we can develop and extend all of the students.

3 The Assignment

Each semester we have an assignment scenario where students are asked to develop a simulation for a "real world" problem. Previous topics have included the spread of disease; parking cars; brine shrimp life cycles; and the game of cats. The assignment is assessed on three major views of the student's work: 30% demonstration of code, 30% implementation of code, and 40% for the reflective report.

Challenge assignments are not new in programming units - they have been a staple from the earliest courses. In [10] the patterns of student performance in coding challenges is explored. These are smaller task than those undertaken in this unit - where we also challenge the student in terms of planning, time management and persistence. Just as the assignment takes around four weeks for the student, they take time to assess. We find each assignment takes 20-30 minutes to mark. With a large class, that may mean 300 hours of marking. Some automation of assessment is possible, which tends to lead to a very defined project where student solutions can be put through a test suite. We believe that there is much to gain from a "loose" specification, allowing for creativity and choices, which are then justified in the submitted report.

3.1 Generating Snowflakes

For the 2022 Summer School, we asked students to simulate the growth of a snowflake (such as in Figure 2); and then generate multiple snowflakes and simulate their movement in space. The inspiration was taken from a video of a scientist who had discovered the secret to creating custom snow flakes [11]. With some additional source material provided [12], students were ready to start thinking about the approach to the problem.

"No two snowflakes are alike." – Wilson Bentley

Table 1 gives the outline of the problem given to the students. It is intentionally open to interpretation, allowing students to take their work in an individual direction. Within the simulation, there are five features that are expected to be implemented and expanded on (Table 2): object behaviour, movement, terrain and boundaries, interaction and visualisation/results. Bonus marks can be gained for utilising a game engine to implement the problem, and for other exceptional approaches e.g. complex interactions, entertaining visualisations, or complex terrain rendering.

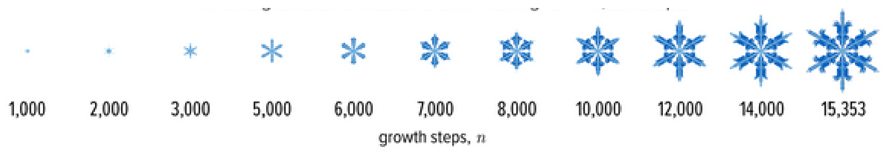


Fig. 2: Snowflake growth [13]

Table 1: Assignment: Problem Definition

The Problem
We will be simulating the generation of snowflakes. Each snowflake will have data as to its shape, then be drawn into a sky/world where it will progress down in a simulated drifting movement. You will be given some sample code, showing a range of approaches to similar problems. For the assignment, you will develop code to model the snowflakes using objects, and to add features to the simulation (e.g. more functionality, statistics, graphics). Your task is to extend the code and then showcase your simulation, varying the input parameters, to show how they impact the overall simulation.

Much of the work for the assignment is to consider the different facets of the problem, and where they might take them. Sample code was given for the sub-problem of taking a quarter of a snowflake and duplicating and flipping it to make the whole snowflake.

At this point in the semester, we have just covered object-orientation (OO), which is a challenging concept for most students. In practicals and tests, the students have seen a range of array-based simulations. Many choose to ignore OO in the assignment, and stay in the comfort zone of arrays. Very high marks are still possible.

The reminder: *'Think before you code!'* is included in the assignment specification. Invariably, students reflect on their assignments and their surprise at the amount of thinking that was required, and how often a challenging idea took a small amount of code (and vice-versa). They also get to see large-scale repetition and the need for functions.

3.2 Scaffolding

Each semester there is an air of bewilderment about where to start with the assignment. To help with the initial steps, we provide some starter code (Table 3). This has typically been a simplistic model of a simulation, with x,y positions or population counts held in arrays. These approaches can be extended for those who find coding difficult, while stronger students will be able to convert to objects to represent each entity. High marks are possible with either approach. For the snowflake assignment, students were given code to take a quarter of a snowflake and flip it three ways to form a symmetrical snowflake twice as high

Table 2: Assignment: Required Extensions

The required extensions are: 1. **Object Behaviour:** Extend to have the snowflakes represented as objects. Each snowflake will have a position (in the overview plot) and its data/shape. Prompts: What makes the snowflake unique in its growth? How will you capture/simulate those factors? 2. **Movement:** Snowflakes should have a movement based on gravity and (simulated) wind. Prompts: Is the "wind" constant or does it vary? Does the "wind" affect all snowflakes uniformly? 3. **Terrain and Boundaries:** You could have a 2d terrain read from a file, including some representation of the height of the terrain. There should be boundaries to stop the snowflakes going beyond the grid. Prompts: How do you stop the snowflakes moving to invalid spaces? What happens when they hit a boundary? 4. **Interaction:** Your code should recognise when snowflakes meet/overlap and respond accordingly. Decide what this does to their movement and to the visualisation. Prompts: Do the snowflakes affect each other's path? [How] will overlaps affect the visualisation? 5. **Visualisation/Results:** Enhance the display for the simulation to show multiple snowflakes moving in a world/sky. You should display a log of events and/or statistics for the simulation and also be able to save the simulation state as a plot image or a csv file. Prompts: How will you differentiate the snowflakes? What useful information could you display? 6. **BONUS:** Utilise a game engine or plotting package to allow interaction with the movement and generation of the snowflakes. Prompts: What interaction could you have? Discuss this in future work if you don't attempt it. Note: Your program should allow command line arguments to control the parameters of the experiment/simulation.

and twice as wide. This was also used within one of the Practical Tests to ensure students were familiar with the code as preparation for the assignment.

In previous semesters, only specific starter code was given. Now we also make a collection of starter code from earlier semesters available. This might seem to be giving too much help, however there is learning in reading other people's code and evaluating its applicability. Students frequently talk about the "gremlin approach" or the "shrimp approach" when they have taken inspiration from the sample code.

This semester we gave more of a walkthrough of the approach to developing the sample code. The code wasn't actually given - it appeared in a screenshot in an announcement. Students could then follow the steps towards the final version of the supplied. We are moving towards providing coding case studies as we feel providing an exemplar of the process of coding is more valuable than just giving access to code that is beyond their level.

Table 3: Help given to students as a starting point

The assignment is now available on the Assessments page. You will have until Friday 18th March to complete it (just over 4 weeks).

There is starter code from previous assignments on the page with the assignment specification. For this assignment, I'm including an approach and some starter code for a snowflake that may be useful:

Making Snowflakes

As it's a very visual problem, you should start with making sure you can plot what you generate. The versions I went through to get to this are:

1. Generate and plot a 2d array of random integers. Have a variable "size" to make it easy to update the array size.
2. Make an array of zeros 2x the #rows and 2x # cols, then slot the array from (1) into the top left of the array
3. Now slot duplicates of the quarter into the other three spaces: top right, bottom left, bottom right. (I multiplied the values by 2,3,4 to make it easier to check it was correct)
4. Modify the slotting code in (3) to flip rows/cols when copying to give symmetry
5. Modify (4) to put the building of the full snowflake into a function - it will be easier to use and make the code more readable. The function should stand alone - so don't reference variables that are outside the function (I changed their names to avoid confusion)

I've put some screenshots in below to show the steps. Hopefully this will give something of a starting point for the assignment, noting that snowflake symmetry is a little different to this - but you only need to generate a quarter, then hand off the duplication to the function.

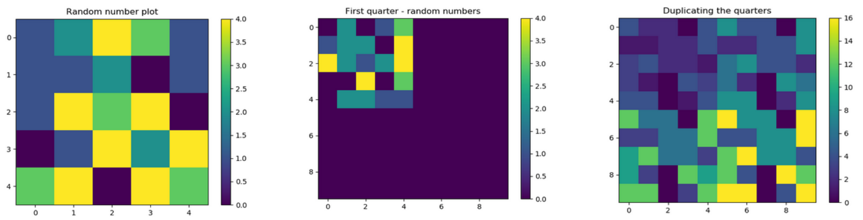
You are welcome to take a different approach! Four or eight pointed snowflakes are OK (and easier), but six is more realistic.

3.3 Reflection

The journey through the assignment, and the coding experience, is different for each student. 40% of the marks for the assignment are allocated to the report, to allow reflection and showcasing of their work - giving maximum chance for letting the marker know what they did and why. A perfect program would only achieve 60% in the unit without the report. The report structure is given in Table4.

In coding assignments, we have noted some patterns in the students' performance, which can be related to the Digital Proficiency Levels in Figure 1:

1. **I don't want to talk about it -** code doesn't go much beyond the sample code combined with snippets from lectures and practicals. Usually accompanied by an apology *(Proficiency: Awareness)*.
2. **I think I'm getting it! -** Has a go at each of the required extensions, may still struggle with object-orientation, so will often just have the x,y position



(a) Version 1 (b) Version 2 (c) Version 3

Fig. 3: Example output of first three versions of snowflake code

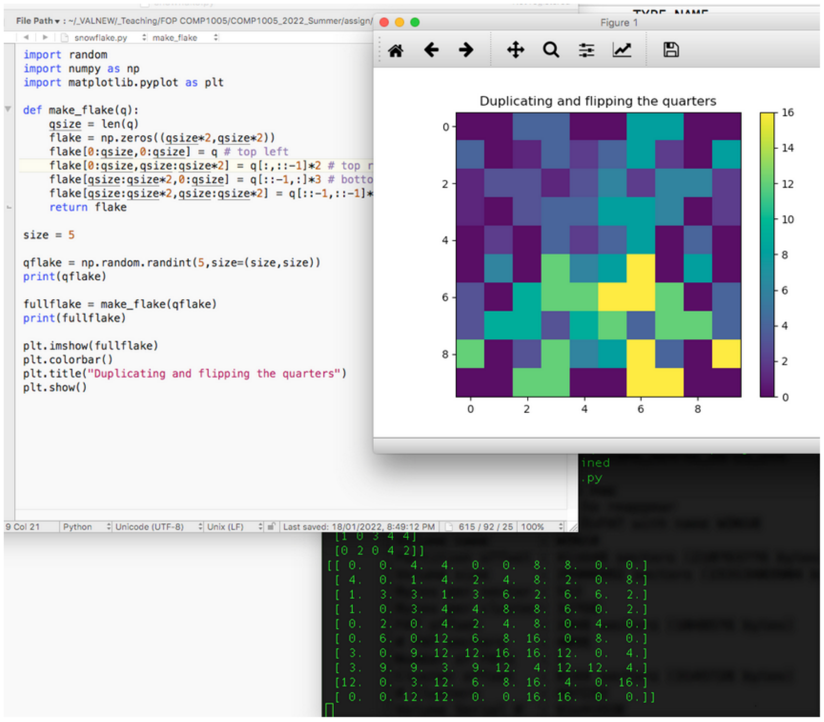
in the class and still be doing most of the work with arrays *(Proficiency: Literacy)*.
3. **Hits the marks** - works to the rubric to have a solid attempt at all requirements. Could get full marks... given more time *(Proficiency: Competence)*.
4. **Exceeds expectations** - brings functionality, complexity and artistry that surprises and excites the markers *(Proficiency: Expertise)*.

In attempting the assignment, students not only need to have some coding skill, they then need to apply that to a simulation. This simulation came in two parts: generating the snowflake(s) and moving the snowflakes through the scene (plot window).

Students in the *Awareness* level would tend to address only the second part, taking the advice that many of the extensions could be attempted without a fancy snowflake generator. These students generated matplotlib.pyplot `scatterplot( )` with circles to represent the snowflakes. Working from the "gremlins" code-base, the snowflakes were generated at random points at the top of the scene, then worked with one of: a speed in the x and y directions; a random movement in the x and y directions or a random choice from a list of possible movements in the x and y directions.

With a bit more understanding (*Literacy*) we began to see the basic simulations be based on simple objects with positions. Plots were still `scatterplot( )` circles moving down the scene, but now there were colormaps and background colours creating a more "snow-like" scene. One other extension would be attempted, usually wind or terrain. With the wind, the snowflakes would have a breeze come through, either uniformly or randomly affecting the snowflakes. The terrain would be represented as blocks, with tests for each snowflake to see if it hit the ground (and disappeared). These students were excited about their work as they had achieved more than they expected!

Students who had gained enough knowledge for *Competence* attempted all extensions and spoke with excitement of the additional work they could have done. Their snowflakes were pixelated images, often based on the supplied code. This required a shift from scatterplots to displaying a compiled image with `imshow( )`, with the 2-D array representation of the snowflake being held in an object, along with its position. These simulations often tested for the terrain


File Path ▾ : ~/_VALNEW/_Teaching/FOP COMP1005/COMP1005_2022_Summer/assign/
snowflake.py
make_flake
Figure 1
Duplicating and flipping the quarters
9 Col 21
Python
Unicode (UTF-8)
Unix (LF)
Last saved: 18/01/2022, 8:49:12 PM
615 / 92 / 25
100%

```
import random
import numpy as np
import matplotlib.pyplot as plt

def make_flake(q):
    qsize = len(q)
    flake = np.zeros((qsize*2,qsize*2))
    flake[0:qsize,0:qsize] = q # top left
    flake[0:qsize,qsize:qsize*2] = q[:,::-1]*2 # top r
    flake[qsize:qsize*2,0:qsize] = q[::-1,:]*3 # botto
    flake[qsize:qsize*2,qsize:qsize*2] = q[::-1,::-1]*
    return flake

size = 5

qflake = np.random.randint(5,size=(size,size))
print(qflake)

fullflake = make_flake(qflake)
print(fullflake)

plt.imshow(fullflake)
plt.colorbar()
plt.title("Duplicating and flipping the quarters")
plt.show()
```

```
 [1 0 3 4 4]
 [0 2 0 4 2]]
[[ 0.  0.  4.  4.  0.  0.  8.  8.  0.  0.]
 [ 4.  0.  1.  4.  2.  4.  8.  2.  0.  8.]
 [ 1.  3.  3.  1.  3.  6.  2.  6.  6.  2.]
 [ 1.  0.  3.  4.  4.  8.  8.  6.  0.  2.]
 [ 0.  2.  0.  4.  2.  4.  8.  0.  4.  0.]
 [ 0.  6.  0. 12.  6.  8. 16.  0.  8.  0.]
 [ 3.  0.  9. 12. 12. 16. 16. 12.  0.  4.]
 [ 3.  9.  9.  3.  9. 12.  4. 12. 12.  4.]
 [12.  0.  3. 12.  6.  8. 16.  4.  0. 16.]
 [ 0.  0. 12. 12.  0.  0. 16. 16.  0.  0.]]
```

Fig. 4: Output of versions (4 and 5) is below, along with the related code

and would create a layer of snow when the snowflake landed. Alternatively, they identified overlapping snowflakes, with print statements to report these "interactions".

The assignments where the students had gained some *Expertise* were delightful. They had taken the time to not only convert to objects, but also to utilise a game engine (which was not covered in the unit). They had a variety of snowflakes, including variation in size and colour. Wind was controlled though interaction with the game, throwing gusts through the snowflake field. Overlapping snowflakes would change colour or change direction.

Living in Western Australia, most of the class has never seen snow, so that may have added another level of curiosity (and challenge) to the assignment.

3.4 Assessment

With many years experience in setting and assessing assignments for Computer Science students, changing the common student attitude that a fully-functional program is all-important has been a bit of a crusade. We want the students to write with good style, to make considered decisions in their approach and

Table 4: Report structure to guide student reflection

Simulation Project Report

You will need to describe how you approached the implementation of the simulation, and explain to users how to run the program. You will then showcase the application(s) you have developed, and use them to explore the simulation outputs. This exploration would include changing parameters, simulation time and perhaps comparing outcomes if you switch various features on/off.
Your Documentation will be around 6-10 pages and should include the following:

1. **Overview** of your program's purpose and features.
2. **User Guide** on how to use your simulation (and parameter sweep code, if applicable)
3. **Traceability Matrix** for each feature, give a short description, state where it is implemented, and how you've tested it (if unfinished, say "not implemented")
4. **Showcase** of your code output, including:
   – Introduction: A discussion of your code, explaining how it works, any additional features and how you implemented them. Explain the features you are showcasing, modifications and parameters you are investigating
   – Methodology: Describe how you have chosen to set up and compare the simulations for the showcase. Include commands, input files, outputs – anything needed to reproduce your results.
   – Results: Present the results of at least three different simulations.
5. **Conclusion and Future Work:** Give conclusions and what further investigations and/or model extensions could follow.

to reflect on their creation and be able to critique their own work. The marks awarded for the submissions are as follows:

– **Code Features.** (30 marks) Based on implementation and documentation.
– **Demonstration.** (30 marks) Students demonstrate their code and respond to questions from the markers.
– **Report.** (40 marks) As described in Table 4.

The bonus of this is the students who are less strong on coding, are often quite comfortable writing a report about their experience. This is also very important for science and engineering students, who's future programs will not be an end in themselves - they will be the basis for decision-making and analysis. For those situations, the ability to discuss their code, their decisions, assumptions and limitations are vital.

3.5 Academic Integrity

The easy accessibility of information and answers has aided programmers incredibly. Unfortunately, that also increases the potential for students to find or source answers which may range from how to use colormaps, through testing for the intersection of objects, to being able to pay someone to do their assignment for them. The more students in the class, the more difficult identifying these issues becomes.

“We’re like snowflakes. Each of us is unique, but it’s still pretty hard to tell us apart.” - Tony Vigorito.

Our most time-consuming, but also most effective, mechanism for assessing academic integrity is the assignment demonstrations. Unless a cheating student has done some exceptional preparation, they will find it difficult to describe code they haven’t written themselves. We also see very different style in the code from students in our course, to code they may have accessed or procured.

“They say that there can never be two snowflakes that are exactly alike, but has anyone checked lately?” - Terry Pratchett.

In a more formal approach, we also have an in-house tool, Tokendiff [14], for comparing student assignments for similarity. This tool provides a report on each pair of assignments to indicate the code that is too similar. It is able to account for rearranged code, and for renamed variables, which is about as far as a student goes if they “need” to copy.

By providing quite obscure problems, following a family of simulations and scripting that is fairly unique, students have indicated there is *nothing* on the Internet to help with their assignments. The challenge is to continue to come up with new ideas!

4 Results

Students took an impressive range of paths with the same starting point for the assignment. Typically we see strains of similar assignments, particularly at the lower end of scores. That was not noticed in this smaller class group, where, beyond the supplied scaffolding code, they went in different directions.

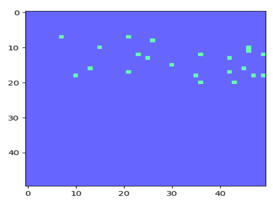
4.1 Variations on a Theme

In line with the proficiency categories, the images below provide a indication of the output of simulations in the students’ assignment work.

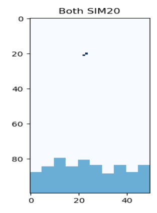
4.2 Reflections and Reactions

“Every avalanche begins with the movement of a single snowflake, and my hope is to move a snowflake.” - Thomas Frey.

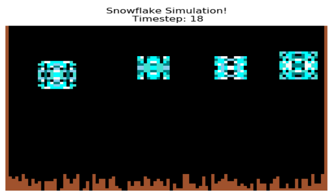
Getting that first snowflake movement was cause for celebration. That was soon followed by a drive to have more or different snowflakes, to adjust their movement or change the colours to be more relatable. Students expressed relief



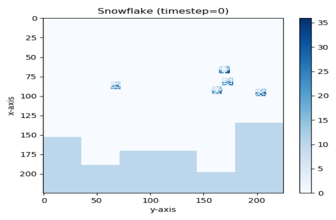
(a) Awareness level



(b) Literacy level



(c) Competence level (sample 1)



(d) Competence level (sample 2)

Fig. 5: Example output of submitted assignments at competency levels 1-3

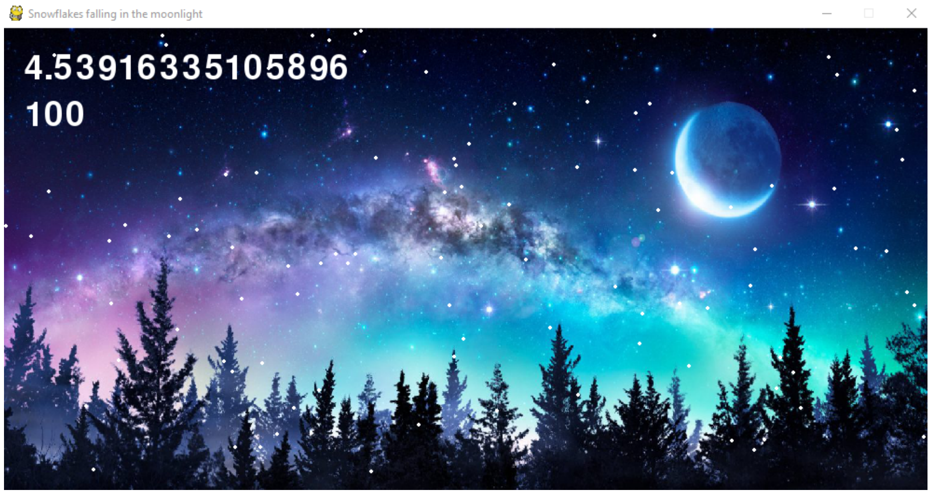
and pride in their coding outcomes, at all levels of proficiency. The subject for the assignment was so approachable, it was possible to show family members their work and share the excitement of the activity (not always possible with university assignments).

As the students moved through the unit, and the assignment, the snowflake took on a greater meaning. Where once it may have been an insult or slur against a generation, they were being creative and taking it as a challenge. The analogy extends further in an education environment, where we might see the students' knowledge and skills growing from a small core. As they extend themselves to new concepts, tentatively at first, then with more strength, their snowflake of understanding grows. It spans from the simple concepts of variables and control structures, then reinforces and extends on those with collections and functions. And this is a viable snowflake, but it can go further: into object-orientation, scripting and automation and beyond. Eventually they follow their unique path of connections and concepts to make their snowflake - similar but different to that of any other student. We, as educators, need to create the right conditions for that growth to happen.

> *"Advice is like snow - the softer it falls, the longer it dwells upon, and the deeper it sinks into the mind."* -Jeremiah Seed.

5 Conclusion

When teaching Fundamentals of Programming, we aim to impart far more than how to put lines of code together. We realise many students find coding daunting



```
4.53916335105896
100
```

Fig. 6: Output at Expertise level

and confronting: and programming languages are not gentle when they find an error! So we work to provide a safe environment and find interesting, relevant and often quirky scenarios to help students overcome their fears and go boldly into the code. We also aim to spark their imaginations to see the potential of simulations, analysis, visualisation and automation.

The assignments are a necessary leap into the unknown, with some scaffolding provided. Students are free to explore their own line of implementation - there is no correct or model solution. For many students, this in itself is unsettling, however we seem to win most of them over by the end of semester. Regardless of the proficiency of the student, they find the assignment challenging, and will usually present something they are proud of, or be able to reflect on what they would have liked to do - which is clearer once they done things the "wrong" way.

The students in the course, reputably part of a "snowflake generation", are initially hesitant. They may be expert users of technology, but coding strips that away to the essence of problems solving without wizards and templates. To move beyond the unforgiving nature of the interpreter, through the bewildering pedantic-ness of code and then confidently put forward a solution to a not-yet-resolved challenge, takes grit and perseverance. Many would back away from this uncomfortable space, but these students push on. They hopefully have become more confident when encountering the unknown, and will take this strength on with them in their future study and careers.

6 Acknowledgements

The author is grateful for the efforts of all students who have undertaken Fundamentals of Programming at Curtin. In particular, the Summer School students

of 2022 were brave enough to try the new format for the unit, and to produce the astonishing work discussed in this paper.

References

1. Forbes Technology Council. Should Everyone Learn To Code? 15 Tech Pros Weigh In On Why Or Why Not. ForbesMay 20, 2020. `https://www.forbes.com/sites/forbestechcouncil/2020/03/20/should-everyone-learn-to-code-15-tech-pros-weigh-in-on-why-or-why-not/?sh=492945a7693e` Last accessed 10 Feb 2022
2. Huggard, M. (2004). Programming Trauma: Can it be Avoided?.
3. CoderDojo. `https://coderdojo.com/` Last accessed 10 Feb 2022
4. Sheridan I., Goggin D. and L. Sullivan. (2016). Exploration of Learning Gained Through CoderDojo Coding Activities. 10.21125/inted.2016.0545.
5. Fletcher A., Mason R., and G. Cooper. 2021. Helping students get IT: Investigating the longitudinal impacts of IT school outreach in Australia. In Australasian Computing Education Conference (ACE '21). Association for Computing Machinery, New York, NY, USA, 115–124. DOI:https://doi.org/10.1145/3441636.3442312
6. Pirie, I. The measurement of programming ability. University of Glasgow (United Kingdom). ProQuest Dissertations Publishing, 1975. 10778059.
7. Wilson G. Software Carpentry: lessons learned [version 2; peer review: 3 approved]. F1000Research 2016, 3:62 (https://doi.org/10.12688/f1000research.3-62.v2)
8. SHODOR: A National Resource for Computational Science Education. `http://www.shodor.org/`. Last accessed 4 Feb 2020
9. Haslam-Ormerod S. 'Snowflake millennial' label is inaccurate and reverses progress to destigmatise mental health. The COnversation. `https://theconversation.com/snowflake-millennial-label-is-inaccurate-and-reverses-progress-to-destigmatise-mental-health-109667` accessed 20/2/2022.
10. Kadar, R., Wahab, N. A., Othman, J., Shamsuddin, M., Mahlan, S. B. (2021). A Study of Difficulties in Teaching and Learning Programming: A Systematic Literature Review. International Journal of Academic Research in Progressive Education and Development, 10(3), 591–605.
11. Veritasium: The Snowflake Mystery - YouTube video. `https://www.youtube.com/watch?v=ao2Jfm35XeE`. Last accessed 10 Feb 2022
12. weather.gov Snowflake Science. `https://www.weather.gov/apx/snowflakescience`. Last accessed 10 Feb 2022
13. Krzywinski M. and J. Lever. In Silico Flurries - Computing a world of snowflakes, December 23, 2017 `https://blogs.scientificamerican.com/sa-visual/in-silico-flurries/` Accessed 19 Feb 2022
14. Cooper D. (2022) TokenDiff – a source code comparison tool to support the detection of collusion and plagiarism in coding assignments. `(https://bitbucket.org/cooperdja/tokendiff`. Last accessed 6 Nov 2022