Design Studio: Algorithms as Future Makers

A modified version [of the assignment created]() by Evan Peck at Bucknell University

In this lab, we will create algorithms that decide who will get an interview

We have seen that algorithms have the potential to assign value to people and make decisions based on those values. Here, we will experiment with that idea in another *very real* context that will likely impact your job prospects in the future.

In this studio, you'll practice...

- For loops
- Processing large quantities of data

Moogle's Hiring Algorithm: Who gets an interview?

Imagine you are working for *Moogle*, a well-known tech company that receives tens of thousands of job applications from graduating seniors every year. Since the company receives too many job applications for the Human Resources Department (HR) to individually assess in a reasonable amount of time, you are asked to create a program that algorithmically analyzes applications and selects the ones most worth passing onto HR. **Your job is to build an algorithm that helps determine the order which job applicants will be called in for an interview.** [This is a real scenario playing out every day and will likely impact you as an applicant someday.]()

Applicant Data

To make it easier for their algorithms to process, *Moogle* designs their application forms to get some numerical data about their applicants' education. Applicants must enter the grades they receive from their 7 core CS courses, as well as their overall non-CS GPA at the university.

To make things simpler, let's think about the 7 courses that you will study at SFSU if you choose to take a CS major:

1. CSC 101: Introduction to Computing

2. CSC 210: Introduction to Object-Oriented Programming (aka Intro to Computer Programming)
3. CSC 220: Data Structures
4. CSC 340: Programming Methodology
5. CSC 413: Software Development
6. CSC 415: Operating Systems
7. CSC 510: Analysis of Algorithms

For your convenience, the grades will be automatically converted to a score and stored in an array that you can access. For example, a student who received the following score (after converting from their grades) ...

- **Intro to Computing:** 100
- **Intro to Object-Oriented Programming**: 90
- **Data Structures:** 95
- **Programming Methodology:** 80
- **Software Development:** 100
- **Operating Systems:** 75
- **Analysis of Algorithms:** 95

... would result in the following list:

```
[100, 90, 95, 80, 100, 75, 95]
```

- You can assume the same order (i.e., index 0 is *always* Intro to Computing, 1 is *always* Intro to Object-Oriented Programming, and so on).

Create the Analysis Algorithms We Specify

In your code, you will be writing a series of predefined methods which apply different criteria to applicants.

We are first going to write basic methods that Moogle has pre-established. These methods will filter candidates based on some rules that Moogle thinks are important for their purposes.

Your goal: Complete the following functions and test them to prove they work.

• `averageAbove85`	accepts applicants that have an average above 85
• `noGradeBelow65`	accepts applicants that have no grade below 65. This includes all grades. *Hint: This is like the **search pattern** (refer to the lecture slides)*
• `atLeastFour85`	accepts all applicants that have at least 4 grades above 85 (their non-CS GPA counts as a grade in this case)

*Hint: This is like the **accumulator pattern**.*

- `upperLevelAbove85` accepts applicants that have an average above 85 *in their upper-level CS courses*.
  Upper-level courses are those with numbers beginning 3 or more. These are the last 4 elements in the array.

To test your code, run the program and individually try out each method as you create them.

Use the attached Java file to write your code. The main method contains some array variables that you can use to test your implementation.

Create your *own* filter

You've now written and seen 4 different selection filters, *it's time to create your own.*

- Decide what you think would be a fairer way of filtering applications.
  You shouldn't just use what we've created so far. Come up with something *better*. Provide you decision and rationale for filtering applications (in English, not code).
  - What criteria(s) do you want to use to filter applications? (Complete this.)
  - Why did you select your criteria? (Complete this.)
  - So far, we have looked at "easy" ways to measure applicants' abilities – just their grades. You may have identified other criteria above. It is also important to think how users will showcase these abilities so that they can be used in the algorithm. Not all the criteria that you have identified above can be filtered by a computing algorithm. Remember, **not everything that you can measure matters and not everything that matters can be measured**. How will an applicant showcase their abilities to meet all criteria that you have listed above? (Complete this.)

Before you Code, Assess the Needs of Your Users

While the list above was *your* take of important criteria to consider. There are **many** more potential criteria to consider if you want to create a **fair** algorithm that takes into account the diverse applicants. Tech companies need to ensure more diverse workforce.

Group Work: Reach out to 3—4 other students. Discuss the various criteria that you all have come. Create a bullet-point list of all the criteria that came up in conversation. Discuss together to rank the criteria and select all that you think is important to consider.
Note: you all may not agree in the criteria to include. Listen to the differing perspectives and evaluate the difference in thoughts. Note them down below.

- Factor 1:
- Factor 2:
- Factor 3:
- Factor 4:
- Factor 5:
- Factor 6:
- ...

Now, you will write your filtering method.
But before writing your code, provide **at least 3** test cases for your method `customFilter`.

`customFilter:`

- An example: Our data will be an array that has the courses listed above along with three other factors: non-CS GPA, has previous volunteering experience, and number of years working in a related job. We will select a candidate if they have average of CS courses above 70, have non-CS GPA above 3.5, and have previously volunteered somewhere. Thus to test, we will call customFilter(`[100, 90, 95, 80, 100, 75, 95, 3.8, 100, 3]`) should return true.

- Test Case 1:
- Test Case 2:
- Test Case 3:

During your creation, keep a couple of things in mind:

- Use block comments to explain what data that you are collecting to filter the applicants.
- Use both block and inline comments to describe what was happening in the program.
- Choose variable names that clearly describe that data that they hold.
- Use spacing to group similar code.

After you write your code, test it with your test cases. Did it work? If not, analyze why your code is not working.

Your Code Works… but is it fair? (Individual)

You should never deploy real code without checking your assumptions.
Your test cases tested your *technical* assumptions, but not *your social* assumptions.

1. Find classmates either inside or outside of the lab.
2. Run your code with them
3. Get feedback on what worked and what didn't?

In particular, you should reflect on…

1. **Which students who apply for jobs are most likely to benefit from your algorithm?**
2. **Which students who apply for jobs are most likely to be penalized by your algorithm?**

Finally add your reflections to this document.

Your Reflection: Tradeoffs

(REPLACE THE FOLLOWING REFLECTION WITH YOUR REFLECTIONS.)

While the filtering algorithm makes sense for companies to manage when there are large number of applicants, but such algorithms do not take into consideration the various facets that are important to consider when hiring. For example, you could have a kid and managing two jobs while studying but you will be filtered but a single student who is good only in coursework will not be filtered. Of course, the first person who is managing so many responsibilities should be considered more carefully! In the future, I would revise my filtering criteria to include more holistic factors, allowing applicants to tell me why they should be considered.

If you're interested…

- Can an algorithm hire better than a human?
  By Claire Cain Miller (The New York Times; you have free subscription via SFSU)
- Amazon Scraps Secret AI recruiting tool that showed bias towards women
  By Jeffrey Dastin (Reuters)