For our first assignment, we will start by completing several exercises using the Hazel programming environment. These exercises will familiarize you with the basics of functional programming, which we will be focused on over the next several weeks.

Hazel is a simple functional environment with a number of editing features to learn the new syntax and make it easier to write type-correct code. It also o ers live feedback as you code regarding tests and how many points you have earned (see below). The Hazel syntax is very similar to that of other functional languages, notably including OCaml, which is the industrial-strength functional language that we will switch to using later this semester. Hazel itself is written in OCaml!

You will submit your solutions to all parts of this exercise using Gradescope (instructions below). Check that you have access to Gradescope, and if you have not used Gradescope before, familiarize yourself with the submission process. Email the sta if you do not have access to Gradescope.

Late Days You may submit up up to 1 business day late using one of your late days. This means that you may submit by 8:00 PM on the next business day (e.g. if Monday, if not a holiday, you may submit by 8:00 PM on Monday). **You cannot use multiple late days for a single assignment.** Your Gradescope submission time will be the ground truth for determining whether or not you used a late day.

Getting Started

You can access the Hazel instance pre-loaded with the exercises for this assignment using your web browser by going to the following URL:
`https://eecs490.github.io/eecs490.org/assignments/eecs490 f24 a1/`

To introduce the system, we have prepared a 22 minute video that you should watch first (and follow along with if you'd like, pausing as necessary):
`https://youtu.be/qAEJ2z_onnE`

This video uses a slightly older version of the Hazel UI but the basic operations should be very similar. Feel free to ask on Piazza if anything is unclear.

Once you have watched the video (and reviewed the material from Lectures 1-4), you are ready to get started with the exercises. There are **8** exercises totaling **100 points**, and they get progressively more difficult. The point totals are shown in Hazel for each exercise and for each component of each exercise.

2 Submission

Here are some things to keep in mind about Hazel:

Hazel saves your progress **locally**, in your web browser's local storage (so you can refresh the page without losing your progress).

Hazel does **NOT** save your work to a server! So to submit your solutions, you must Export your submission to a file and then upload it to the **A1 Code Submission** assignment on Gradescope. The video shows how to export.

The autograder on Gradescope does not actually grade your code, it merely checks that you submitted a Hazel export le!

We recommend periodically exporting even before you are finished, e.g. after you have made some progress on an exercise, in case you encounter a bug. You can import your previously exported data from the sidebar. Because your data isn't being uploaded to a server, we cannot recover lost work, nor is losing data an excuse not to complete the assignment on time (even if it is due to a Hazel crash or bug), so it is your responsibility to be exporting regularly just in case! We recommend saving your exports to a cloud storage location, like the Google Drive the university provides you.

Do **NOT** open multiple instances of Hazel on the same machine, as they will interfere with one another and you may lose work. If you want to move between di erent machines or browsers, you can use the export and import functionality.

Hazel was most extensively tested on up-to-date Chrome or Chromium-based browsers (Opera, Brave, Edge, etc.) and Firefox. If you are having trouble with it loading at any point this semester, try switching to such a browser.

Getting 100% of the points according to Hazel's autograder does not mean you will get 100% of the points after we grade your submissions.

We will be manually checking that you have followed any instructions that Hazel does not automatically test for.

We will also be docking points for trivial test cases (e.g. `test true en`), duplicate test cases, for solutions that are excessively complex, and for obviously bad style (e.g. you write an entire long function all on one line).

We won't take o points for small issues like variable names or whitespace inconsistencies.

Here are some things to avoid:

Unnecessary use of helper functions

Excessive base cases when pattern matching:

```
case n
    | 0 => 0
    | 1 => 1
    | 2 => 1
    | 3 => 2
    | _ => ...
end
```