# Project - CSE 105 Spring 2022

CSE105Sp23

Due 6/1/23 at 5pm (no penalty late submission until 8am next day)

The project component is designed for you to go deeper and extend your work on assignments and to explore an application of your choosing. The project is an individual assignment and has two tasks:

- Task 1: A meaningful language (written) and
- Task 2: A helpful function (some programming, presented as a screencast video).

**What resources can you use?** This project must be completed individually, without any help from other people, including the course staff (other than logistics support if you get stuck with screencast).

You can use any of this quarter's CSE 105 offering (notes, readings, class videos, homework feedback). These resources should be more than enough.

If you are struggling to get started and want to look elsewhere online, you must acknowledge this by listing and citing any resources you consult (even if you do not explicitly quote them), including any large-language model style resources. Link directly to them and include the name of the author / video creator, any search strings or prompts you used, and the reason you consulted this reference.

The work you submit for the project needs to be your own. Again, you shouldn't need to look anywhere other than this quarter's material and doing so may result in definitions or notations that conflict with our norms in this class so think carefully before you go down this path.

**Submitting the project** You will submit a PDF for Task 1 and a video for Task 2 to an online assignment on Gradescope.

## Task 1: A meaningful language

Automaton models are useful for concisely representing patterns. In this question, you'll choose a pattern in an application you care about, define it precisely, and then build a DFA, NFA, or PDA that recognizes it.

First, pick **one** application for your example. Here are some ideas to get you started - but you can choose to go in a different direction all together.

- Data validation for input in text files (e.g. emails with specific domains, dates in specific formats, PIDs in a class list, etc.)
- Finding ASCII codes for punctuation in a binary file.
- The CDC recommended procedure for hand washing (Refer to the guidelines from the CDC here https://www.cdc.gov/handwashing/index.html in your explanation. You might find the first example in chapter 1 about automatic door controllers helpful when starting your design.)
- See more ideas here:

  `https://theory-cs.github.io/files/practical-applications-of-theory-of-computation.pdf`

Then:

1. In a paragraph or so, give the context for your chosen application and why you chose it.
2. Specify the alphabet for your example.
3. Write a precise (mathematical and/or English) description of a set of strings over this alphabet that is important, and include a sentence or so justifying why this set is important.
4. Give one example of a string in this set and a string not in this set, and explain why you chose these example strings.
5. Classify your language as regular or context-free, and prove this classification by giving an appropriate machine that recognizes your language. Justify your construction.

**Grading criteria and checklists** Solution is typed out in detail step-by-step, with clear and correct logic and justification.

Each of the five items are included, with precise language and notation for all terms and complete, correct, and clear justification.

## Task 2: A helpful function

To relate the difficulty level of one language to another we use mapping reduction, which relies on the notion of computable function. In this part of the project, you will define, program, and trace a specific computable function from $\{a, b\}^*$ to $\{a, b\}^*$.

First, choose a function you will be working with. You can pick any function you like so long as:

- Its domain is $\{a, b\}^*$ and its codomain is $\{a, b\}^*$
- There is at least one domain element that is mapped to a string that is **longer** than it and there is at least one domain element that is mapped to a string that is **shorter** than it.

Then:

1. Give a high-level description of a Turing machine that computes this function.
2. Draw the state diagram of this Turing machine.
3. Write a program in Java, Python, JavaScript, or C++ (or another programming language of your choosing) that simulates the computations of this Turing machine. Your program should display a snapshot of each step of the computation, including the state of the machine, the current (non-blank) contents of the tape, and the location of the read/write head of the Turing machine. If you would like, you may use aids such as co-pilot or ChatGPT to help you write this program. However, you should test the code that is produced and be able to explain what it is doing. As a header in your code file, include a comment block describing any resources that were used to help generate your code.
4. Run your program on a string over $\{a, b\}$ which the function maps to a string that is longer than it, and
5. Run your program on a string over $\{a, b\}$ which the function maps to a string that is shorter than it.

Presenting your reasoning and demonstrating it via screenshare are important skills that also show us a lot of your learning. Getting practice with this style of presentation is a good thing for you to learn in general and a rich way for us to assess your skills. To demonstrate your work, you will create a 3-5 minute screencast video with the following components:

- Start with your face and your student ID for a few seconds at the beginning, and introduce yourself audibly while on screen. You don't have to be on camera for the rest of the video, though it's fine if you are. We are looking for a brief confirmation that it's you creating the video and doing the work you submitted.

- Present the function you will be working with. Your video should include a clear and precise definition of the function (before you introduce any Turing machines).
- Show on the screen and explain the high-level description of your Turing machine witnessing that your function is computable.
- Show on the screen and explain the state diagram of your Turing machine.
- Show on the screen and present your code, including the software design choices you made (e.g. which data structures are you using, etc.) and any resources you used.
- Demonstrate running your code on the two example input strings specified above. Explain why the output of your program is what you would expect.

**Grading criteria and checklists** Logistics: video loads correctly, is between 3 and 5 minutes, shows the student's face and ID, and they introduce themself audibly while on screen.

The video clearly states which function was chosen for study, the function which is well-defined and computable, the video presents the two different descriptions of the Turing machine clearly, and the Turing machine correctly computes the function.

The video clearly describes which programming language was chosen for the implementation and gives the reasons why.

The video discusses the connections between the state diagram of the Turing machine and its implementation in the code.

The video clearly demonstrates all test cases, including both expected and actual output. The video should include screencasts of running the code live to demonstrate these test cases.

**Your video:** You may produce screencasts with any software you choose. One option is to record yourself with Zoom; a tutorial on how to use Zoom to record a screencast (courtesy of Prof. Joe Politz) is here:

`https://drive.google.com/open?id=1KROMAQuTCk4OzwrEFotlYSJJQdcG_GUU`.

The video that was produced from that recording session in Zoom is here:

`https://drive.google.com/open?id=1MxJN6CQcXqIbOekDYMxjh7mTt1TyRVMl`

Please send an email to the instructors (minnes@ucsd.edu and dgrier@ucsd.edu) if you have concerns about the video / screencast components of this project or cannot complete projects in this style for some reason.