**Top Down Design**

**Olivia, Claudia, Peter, Benjamin**

Ascii Clock Planning

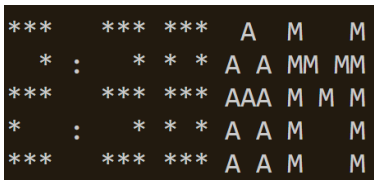
- Prepare digits
    - Create dictionary with lists of strings for each digit
- Input from user:
    - Three inputs, time, clock type, and preferred character
    - Confirm input is valid
        - If input is not valid, tell user character is not permitted and to select another character
- Parse input depending on clock type
    - If a 12 hour clock is chosen
        - If the number entered is larger than 12:00
            - Subtract 12 from the hour and display PM
        - If hour is 12 return 12 + minutes + PM
        - If hour is 0 return 12 + minutes + AM
        - Otherwise attach an AM to string
    - If a 24 hour clock is chosen
        - Keep it the same as the input
- Display time with proper white space and colons
    - Loop from row 1 - 5
        - Loop through each digit in provided string
            - Determine correct character depending on user input
                - If no character is given, use the digit's character
                - Don't change character if :, A, P, or M
            - Modify corresponding list with correct character
            - Add current row of digit into row_output
            - Print row_output, joined by spaces

**Variables**

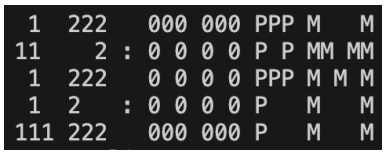
- dictionary = lists of lists of strings representing each digit and letter possible
- permitted = a-z, @, $, &, *, =: permitted characters to display time with; str
- time = input(): time entered by the user; str
- clock_type = input(): number entered by user to indicate to use a 12 or 24 clock; str
- character = input(): character entered by the user, check if permitted before moving on; str
- current_row: row n of the current digit to be replaced with the corresponding character and printed
- row_output: string representing an entire row of the clock, which is to be concatenated upon as each digit is looped through
- character_used: actual character used depending on the current digit being printed

**Test Cases**

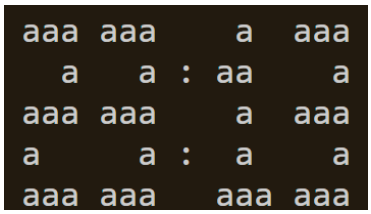
1. 2:38, 12, *, normal




2. 12:00, 12, edge

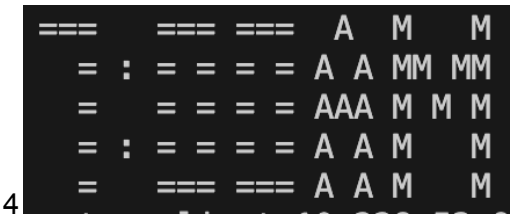



3. 23:13, 24, a, edge

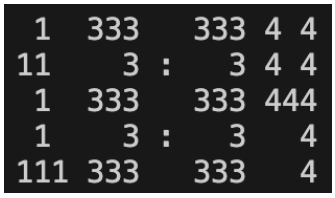



4. 0:19, 24, $, edge


5. 07:00, 12, =, normal




6. 13:34, 24, normal




**Part G**

- Describe the difficulty your team had when combining the code at the end. How could your team have specified the design more clearly at the beginning?

One problem our team had when combining code was that we did not have everything needed for the program to run properly. For example, we needed additional if-statements in the “parse input depending on clock type” part that we did not think about before running the code in ZyBooks. We also encountered slight consistency issues in the code, for example. The ASCII letters were initially defined in a list, while the rest of the code only worked if it was defined as a dictionary.

- Describe any benefits and drawbacks you saw by dividing the coding like this. Can you see reasons why this might be a good idea? How about a bad idea?

A potential benefit of dividing the code is that it reduces the workload of each individual person. In addition, you can easily break down tasks into smaller parts and get work done more quickly. The biggest issue we faced is not being able to see the rest of the project. You are also unable to easily test your part of the code without the other people’s parts, which would defeat the purpose of splitting the work.