Jerry Cain
CS106AX

Handout #13S
October 23, 2026

Solution for Section #4

Solution 1: Roman Numerals [courtesy of Eric Roberts]

```
/*
 * Function: romanToDecimal
 * ------------------------
 * Traverses what we'll assume to be a well-formed Roman numeral
 * and returns the decimal equivalent.
 */
function romanToDecimal(numeral, symbolToValueMap) {
   let result = 0;
   for (let i = 0; i < numeral.length; i++) {
      let currentSymbol = numeral[i];
      let currentValue = symbolToValueMap[currentSymbol];
      if (currentValue === undefined) return -1; // improper numeral
      if (i < numeral.length - 1) {
         let nextSymbol = numeral[i + 1];
         let nextValue = symbolToValueMap[nextSymbol];
         if (nextValue === undefined) return -1; // improper numeral

         if (currentValue < nextValue) {
            result += nextValue - currentValue;
            i++; // Skip next letter
         } else {
            result += currentValue;
         }
      } else { // on last symbol? no next symbol!
         result += currentValue;
      }
   }

   return result;
}
```

Solution 2: Scaling Recipes

```
/**
 * Function: findUnitOfMeasure
 * ---------------------------
 * Returns the index of the record within CONVERSIONS that houses
 * information about the named unit of measure.
 */
function findUnitOfMeasure(unit) {
   for (let i = 0; i < CONVERSIONS.length; i++) {
      if (CONVERSIONS[i].unit === unit) return i;
   }
   return -1; // unit of measure isn't one to be normalized
}

/**
 * Function: normalize
 * -------------------
 * Adjusts the supplied ingredient record so that the quantity
 * is expressed in the most appropriate unit of measure.
*/
function normalize(ingredient) {
   let index = findUnitOfMeasure(ingredient.unit);
   if (index === -1) return;
   for (let i = index - 1; i >= 0; i--) {
      ingredient.amount *= CONVERSIONS[i].amount;
   }
   ingredient.unit = CONVERSIONS[0].unit;
   for (let i = 0; i < CONVERSIONS.length - 1; i++) {
      if (ingredient.amount < CONVERSIONS[i].amount) break;
      ingredient.amount /= CONVERSIONS[i].amount;
      ingredient.unit = CONVERSIONS[i + 1].unit; // advance to higher unit
   } // bound is length - 1, as each iteration accesses i and i + 1
}

/**
 * Function: scale
 * ---------------
 * Scales the recipe so that the quantities expressed are enough for
 * precisely 'servings' servings.
 */
function scale(recipe, servings) {
   let scale = servings/recipe.servings;
   for (let i = 0; i < recipe.ingredients.length; i++) {
      recipe.ingredients[i].amount *= scale;
      normalize(recipe.ingredients[i]);
   }
   recipe.servings = servings;
}
```

Solution 3: Enigma Reflector Construction

```
/**
 * Function: buildReflector
 * ------------------------
 * Builds and returns a random reflector by pairing letters of the
 * uppercase alphabet. Each letter maps to its partner, so no letter
 * maps to itself and reflecting twice recovers the original letter.
 * The character at each index gives the reflection of the letter
 * at that same index in ALPHABET.
 */

function buildReflector() {
   let reflector = STARTER;
   let candidates = ALPHABET;

   while (candidates.length > 0) {
      let pos1 = randomInteger(0, candidates.length - 1);
      let ch1 = candidates.charAt(pos1);
      candidates =
         candidates.substring(0, pos1) + candidates.substring(pos1 + 1);

      let pos2 = randomInteger(0, candidates.length - 1);
      let ch2 = candidates.charAt(pos2);
      candidates =
         candidates.substring(0, pos2) + candidates.substring(pos2 + 1);

      let index1 = ALPHABET.indexOf(ch1);
      let index2 = ALPHABET.indexOf(ch2);
      reflector =
         reflector.substring(0, index1) + ch2 + reflector.substring(index1 + 1);
      reflector =
         reflector.substring(0, index2) + ch1 + reflector.substring(index2 + 1);
   }

   return reflector;
}
```

Solution 4: Move-To-Front Encryption

```
/**
 * Function: encrypt
 * -----------------
 * Encrypts a word using a key that begins as the lowercase alphabet.
 * Each letter is encoded as the alphabet character at its current
 * index in the key, then moved to the front of the key while leaving
 * all other letters in their original relative order.
 * Assumes the word contains only lowercase English letters.
 */

function encrypt(word) {
   let key = ALPHABET;
   let encryption = "";
   for (let i = 0; i < word.length; i++) {
      let ch = word[i];
      let index = key.indexOf(ch);
      encryption += ALPHABET.charAt(index);
      key = ch + key.substring(0, index) + key.substring(index + 1);
   }
   return encryption;
}
```