Jerry Cain
CS 106AX

Handout #14
October 25, 2023

Section Handout #4—More Data Structures, Python

Problem 1: Roman Numerals (in JavaScript)

In Roman numerals, characters of the alphabet are used to represent integers as shown in this table:

symbol	value
`I`	1
`V`	5
`X`	10
`L`	50
`C`	100
`D`	500
`M`	1000

Each character in a Roman numeral stands for the corresponding value. Ordinarily, the value of the Roman numeral as a whole is the sum of the individual character values given in the table. Thus, the string "**`LXXVI`**" denotes 50 + 10 + 10 + 5 + 1, or 76. The only exception occurs when a character corresponding to a smaller value precedes a character representing a larger one, in which case the value of the first letter is subtracted from the total, so that the string "**IX**" corresponds to 10 − 1, or 9.

Write a function **`romanToDecimal`** that takes a string representing a Roman numeral and returns the corresponding decimal number. To find the values of each Roman numeral character, your function should look that character up in a map that implements the Roman numeral conversion. If the string contains characters that are not in the table, **`romanToDecimal`** should return −1.

Problem 2: Scaling Recipes (in JavaScript)

I've decided to throw the biggest of big dinner parties this coming weekend, and I've further decided to cook everything myself. I know what I'm cooking, and I've even converged on the collection of recipes I'll make. Because all of you are invited, you're offering to write a JavaScript program to scale up recipes so I know precisely how much of each ingredient to buy at the supermarket.

Take for example this lovely soup dish I found online, coincidentally expressed in JSON:

```
let soup = {
   name: "Charred Cauliflower Stew",
   servings: 4,
   ingredients: [
      { amount: 2, unit: "head", name: "cauliflower"},
      { amount: 2, unit: "teaspoon", name: "olive oil"},
         ... many more ingredients omitted
      { amount: 1, unit: "cup", name: "watercress"},
   ],
   instructions: [
      "Heat the grill to high and lightly oil the grates.",
      "Toss cauliflower florets with half of the olive oil.",
      ... many more instructions omitted
      "Garnish with remaining watercress and serve."
   ]
};
```

The problem? If I need to cook for 16 people instead of 4, I'd prefer the recipe be structured as follows:

```
{
   name: "Charred Cauliflower Stew",
   servings: 16,
   ingredients: [
      { amount: 8, unit: "head", name: "cauliflower"},
      { amount: 1.33, unit: "ounce", name: "olive oil"},
         ... many more ingredients omitted
      { amount: 1, unit: "quart", name: "watercress"},
   ],
   instructions: [
      "Heat the grill to high and lightly oil the grates.",
      "Toss cauliflower florets with half of the olive oil.",
      ... many more instructions omitted
      "Garnish with remaining watercress and serve."
   ]
};
```

Since the recipe was quadrupled, 2 teaspoons became 8 teaspoons and 1 cup became 4 cups. But those quantities are better expressed in larger units of measure, which is why 8 teaspoons isn't 8 teaspoons or even 2.66 tablespoons, but rather 1.33 ounces. When a quantity can be expressed in terms of 1.0 or more of the next larger unit of measure, it is.

If instead of a party of 16 I throw a party for 1600 (**`#lifegoals`**), I'll need a drastically reformed ingredient list for the soup, as with:

```
{
   name: "Charred Cauliflower Stew",
   servings: 1600,
   ingredients: [
      { amount: 800, unit: "head", name: "cauliflower"},
      { amount: 1.042, unit: "gallon", name: "olive oil"},
         ... many more ingredients omitted
      { amount: 3.125, unit: "bushel", name: "watercress"},
   ],
   instructions: [
      "Heat the grill to high and lightly oil the grates.",
      "Toss cauliflower florets with half of the olive oil.",
      ... many more instructions omitted
      "Garnish with remaining watercress and serve."
   ]
};
```

These conversions are possible, provided we have an ordered list of conversions for each of the standard units of measures. Fortunately, we do, because the internet has everything.

```
const CONVERSIONS = [
   { unit: "dram", amount: 4/3 },
   { unit: "teaspoon", amount: 3 },
   { unit: "tablespoon", amount: 2 },
   { unit: "ounce", amount: 8 },
   { unit: "cup", amount: 2 },
   { unit: "pint", amount: 2 },
   { unit: "quart", amount: 4 },
   { unit: "gallon", amount: 2 },
   { unit: "peck", amount: 4 },
   { unit: "bushel", amount: 55/7 },
   { unit: "barrel", amount: 6000},
   { unit: "acre" } // no larger unit of measure, so no amount field
];
```

Each object in the array constant stores the number of a particular unit needed to make up exactly one of the next larger unit. Therefore, 3 teaspoons make up one tablespoon, 2 tablespoons make up one ounce, 8 ounces make up one cup, and so forth.

Your job as a section it to implement the **`scale`** function, which accepts a recipe object (structured as those you've seen above) and the desired number of servings, and modifies the recipe object the house the amounts needed to serve those many people. If an ingredient's unit of measure appears in **`CONVERSIONS`**, the quantity and unit of measure should be normalized as they have been in the above examples. You must allow for the possibility the supplied recipe needs to be scaled up or down. Hint: convert **all** quantities to drams and then normalize up to the most sensible unit of measure.

```
const CONVERSIONS = [
   { unit: "dram", amount: 4/3 }, { unit: "teaspoon", amount: 3 },
   { unit: "tablespoon", amount: 2 }, { unit: "ounce", amount: 8 },
   { unit: "cup", amount: 2 }, { unit: "pint", amount: 2 },
   { unit: "quart", amount: 4 }, { unit: "gallon", amount: 2 },
   { unit: "peck", amount: 4 }, { unit: "bushel", amount: 55/7 },
   { unit: "barrel", amount: 6000}, { unit: "acre" }
];

function scale(recipe, servings) {
```

Problem 3: Transitioning to Python

Visit the course website at **http://cs106ax.stanford.edu** and drill through the new button along the top labeled **PyCharm**. Follow the instructions to download the most recent version of Python available to us at the moment and a Python development environment called PyCharm, which we referenced during our first Python lecture.

Then implement a Python function that returns the longest palindrome that can be formed by concatenating some nonempty substring of one word (we'll call is **str1**) to some nonempty substring of a second word (we'll call it **str2**). To help guide you, we're presenting the equivalent JavaScript solution. (There's a more elegant solution that compiles arrays of all substrings, but we don't formally know Python lists yet, so we're working with a quadruple **for** loop for one of the few times in my 27 years of teaching.)

Visit the course web page again and drill through the Sections button to arrive at a page that lists all of the section handouts, including this one for Week 5. Download the provided starter code, unzip, and behold subfolders called JavaScript (which contains a fully operational program employing the solution presented below) and Python (which you'll load into PyCharm and modify to emulate the functionality of the JavaScript solution).

Once you implement the Python version of **longestPalindrome** (and, most likely, a helper function called **isPalindrome**), you can use the Terminal within PyCharm to test your program. If all goes well, you should generate output that looks like this (what I type is in bold, and what the program generates is italicized):

```
Jerrys-MacBook-Pro:Python jerry$ python3 palindrome.py bac bac
The longest palindrome is "bab".
Jerrys-MacBook-Pro:Python jerry$ python3 palindrome.py aba aba
The longest palindrome is "abaaba".
Jerrys-MacBook-Pro:Python jerry$ python3 palindrome.py jdfh fds
The longest palindrome is "dfhfd".
Jerrys-MacBook-Pro:Python jerry$ python3 palindrome.py abc def
The longest palindrome is "".
Jerrys-MacBook-Pro:Python jerry$
```

Here's the JavaScript solution you should use to guide your Python implementation.

A few things: **`str.substring(a, b)`** in JavaScript is equivalent to **`str[a:b]`** in Python, and **`str.length`** in JavaScript is the same as **`len(str)`** in Python. We'll learn more about Python strings very soon, but the point of this exercise isn't to learn Python so much as it is to learn the PyCharm development environment.

```
/*
 * Function: longestPalindrome
 * ---------------------------
 * Computes the longest palindrome that can be formed by
 * concatenating some nonempty substring of str1 to some
 * nonempty substring of str2.  The function avoids using
 * arrays, and that necessitates the use of a quadruple for loop.
 */
function longestPalindrome(str1, str2) {
   let longest = "";
   for (let lh1 = 0; lh1 < str1.length; lh1++) {
      for (let rh1 = lh1 + 1; rh1 <= str1.length; rh1++) {
         for (let lh2 = 0; lh2 < str2.length; lh2++) {
            for (let rh2 = lh2 + 1; rh2 <= str2.length; rh2++) {
               let substr1 = str1.substring(lh1, rh1);
               let substr2 = str2.substring(lh2, rh2);
               let candidate = substr1 + substr2;
               if (isPalindrome(candidate) &&
                   candidate.length > longest.length) {
                  longest = candidate
               }
            }
         }
      }
   }
   return longest;
}

/*
 * Function: isPalindrome
 * ----------------------
 * Returns true if and only if the supplied string of lowercase letters
 * is a palindrome.
 */
function isPalindrome(str) {
   for (let i = 0; i < str.length/2; i++) {
      if (str.charAt(i) != str.charAt(str.length - i - 1)) {
         return false;
      }
   }
   return true;
}
```