

Practice Final Examination Solution

Review session: Sunday, December 7th, 12:00 – 2:00 P.M., CoDa B90

Scheduled final: Monday, December 8th, 8:30 – 11:30 A.M., CoDa B90

Solution 1—Python Strings

```
def lrs(s):
    """
    Extracts every possible nonempty substring from the one
    provided and sees if each is repeated a second time.
    While doing so, we keep track of the longest such
    substring and ultimately return that substring at the end.
    """
    longest = ""
    for lh in range(0, len(s)):
        for rh in range(lh + 1, len(s)):
            subs = s[lh:rh]
            if len(subs) > len(longest) and s[lh + 1:].find(subs) != -1:
                longest = subs
    return longest
```

Solution 2—Python Strings and Lists

```
def sort_string(s): return "".join(sorted(s))

def group_anagrams(strings):
    """
    Returns a list of anagram groups as described on the
    previous page.
    """
    groups = []
    for string in strings:
        first = True
        for group in groups:
            if sort_string(string) == sort_string(group[0]):
                first = False
                group.append(string)
        if first: groups.append([string])
    return groups
```

Solution 3—Working with Python Dictionaries and Objects

```
def printCheatSheet(objects, rooms):
    """
    Analyzes the specific dictionary of objects (keys are object names,
    values are instances of AdvObject) and the dictionary of rooms (keys
    are room names, values are instances of AdvRoom) and publishes a little
    cheat sheet to the console, as per the problem specification.
    """
    for objName in objects:
        obj = objects[objName]
        objDesc = obj.getDescription()
        objLocation = obj.getInitialLocation()
        if objLocation != "PLAYER":
            objLocation = rooms[objLocation].getShortDescription()
            print("{} ({} ) starts: {}".format(objName, objDesc, objLocation))
            for roomName in rooms:
                room = rooms[roomName]
                for passage in room.getPassages():
                    if passage[2] == objName:
                        verb = passage[0]
                        desc = room.getShortDescription()
                        print("  Needed for {} from {}".format(verb, desc))
```

Solution 5—Client-Side JavaScript [Solution 4 on next page]

```
/*
 * Function: fetchAndLoadImages
 * -----
 * Issues an asynchronous request to fetch the names
 * and image URLs of all of the users. We ignore the
 * names, but use the image URLs to construct a small
 * HTML component with all those images laid down side-by-side.
 */
function fetchAndLoadImages() {
    AsyncRequest("/api/images")
        .setSuccessHandler(loadImages)
        .send();
}

/*
 * Function: loadImages
 * -----
 * Invoked when GET request initiated by fetchAndLoadImages is
 * issued and the server has responded.
 */
function loadImages(response) {
    let images = JSON.parse(response.getPayload());
    let div = document.getElementById("user-images");
    for (let i = 0; i < images.length; i++) {
        let img = document.createElement("img");
        img.setAttribute("src", images[i].url);
        img.setAttribute("alt", images[i].name);
        img.classList.push("thumbnail");
        div.appendChild(img);
    }
}
```

Solution 4—Defining Python Classes and Reading Files

```

class ElectionData:
    """
    Defines a new type that understands how to read a flat-text
    file of election data and surface information about the
    various British government system's constituencies.
    """
    def __init__(self, filename):
        """
        Constructs the ElectionData object from the information
        provided in the named file, as per the problem description.
        The implementation, as permitted, assumes the file exists and
        that its contents are perfectly formatted.
        """
        self._constituencyNames = []
        self._results = {}
        with open(filename) as infile:
            while True:
                constituency = infile.readline().strip()
                if constituency == "": break
                self._constituencyNames.append(constituency)
                results = []
                while True:
                    line = infile.readline().strip()
                    if line == "": break
                    entry = {}
                    opos = line.find("(")
                    cpos = line.find(")")
                    entry["candidate"] = line[:opos].strip()
                    entry["party"] = line[opos + 1: cpos].strip()
                    entry["votes"] = int(line[cpos + 1:].strip())
                    results.append(entry)
                self._results[constituency] = results

    def getConstituencyNames(self):
        """
        Returns the list of constituencies.
        """
        return self._constituencyNames

    def getResults(self, constituency):
        """
        Returns the election results for the named constituency
        as an array of aggregates, as outlined in the problem
        statement.
        """
        return self._results.get(constituency, [])

```