Cornell NetID, all caps: _____

CS 1110 Regular Prelim 2 Solutions April 2021

1. [14 points] **For-Loops**

   Implement the following function so that it obeys its specification, making effective use of for-loops.

```
def generateStringPairs(x1, x2):
    """
    Returns: list *WITHOUT* duplicates of strings where the first part
    of each string is an element of x1 and the second part is an element of x2.

    Ex: x1 = ["a", 2, "a", "cat"], x2 = [17, "dog"]
        output = ["a17", "adog", "217", "2dog", "cat17", "catdog"]

    Preconditions (no need to assert these):
    x1 and x2 are non-empty lists of strings AND/OR ints (possibly mixed)

    You may NOT use the set() function (if you even know what it is).
    """

    output = []
    for item1 in x1:
        i = str(item1)
        for item2 in x2:
            result = i + str(item2)
            if not result in output:
                output.append(result)
    return output
```

   The specification states that the output list should not have duplicates. It is not enough to remove/ignore duplicates in the input lists. Example: if the two input lists were `["c", "ca"]` and `["at", "t"]`, we would want `["cat", "ct", "caat"]`, not `["cat", "ct", "caat", "cat"]`.

2. **Testing/analyzing recursion.** The code below might not correctly implement its specification.

```
def sum_evens(nums):
    """Returns the sum of all the even numbers in `nums` (returns 0 if there
    aren't any).

    Examples:
        sum_evens([])  should return 0
        sum_evens([5, 11])  should return 0
        sum_evens([2, 11]) should return 2

    Precondition: `nums` is a (possibly empty) list of positive ints.
```

```
    """
    if len(nums) <= 1:
        return 0
    elif nums[0] % 2 == 1:
        return sum_evens(nums[1:])
    else:
        return nums[0] + sum_evens(nums[1:])
```

(a) [7 points] Give a length-3 list of 3 distinct positive ints, at least one of which is even, for which the code above returns the right answer.

[5, 2, 1]

(b) [1 point] If your list is given to the code above as input, what value does the code return?

2

(c) [8 points] Is there a length-3 list of 3 distinct positive ints, at least one of which is even, for which the above code returns the **wrong** answer?

If so,

- write down such a list;
- state what the result of the code *should* be for that list; and
- state what the result *actually is*.

If not, just write the phrase "No such list."

Hint: think through what the base case(s) for this problem should be; use your reasoning when thinking about length-3 lists.

[5, 1, 2] (Notice this is just a reordering of the list above where an even number is the last one in the list). Answer should be 2, but it is 0.

No explanation was needed, but here is the general issue: the case of an input [e] where e is any even number is not being handled correctly, with the effect that if the last entry in the input list is even, it is ignored.

3. [16 points] **Recursion.** Implement the following function, making effective use of recursion.

```
def subcat_names(c):
    """Returns a list of all the names of subcategories for category `c`.
    (Order doesn't matter. Do NOT include the name of `c` itself.

    A category is a list of two items:
        * a non-empty string, like "Animal".
        * a (possibly empty) list of subcategories.

    Example:
        cdh = ["Dwarf Hamster", []]     means subcat_names(cdh) --> []
        h = ["Hamster", [cdh]]          means subcat_names(h) --> ["Dwarf Hamster"]
        fi = ["Fish", []]
        fo = ["Fowl", []]
        a = ["Animal", [fi, fo, h]]     means subcat_names(a) -->
                                         [Fish", "Fowl", Hamster", "Dwarf Hamster"]
```

```
You don't need to know this, but a full listing of `a` is:
['Animal', [['Fish', []], ['Fowl', []], ['Hamster', [['Dwarf Hamster', []]]]]].

if c[1] == []:
    return []
out = []
for subc in c[1]:   # subc[0] = name, subc[1] = list of subcs
    out.append(subc[0])
    out += subcat_names(subc)
return out
```

4. **Implementing methods.** You will write methods for class Shell.

   Some of these methods involve creating/handling File objects. The specification for class File and its `__init__` method are on page 6.

   (a) [5 points] Implement the `__init__` method for class Shell.

```
class Shell:
    """
    Objects represent an instance of a command shell
    (think Terminal for MacOS or Powershell on Windows).

    Attributes:
        homeDir [File]: the home directory for the person using this Shell.
        openDir [File]: the directory currently open in the Shell.
    """

    def __init__(self, home):
        """
        Creates a new Shell with attributes set as follows:
            homeDir: set to `home`
            openDir: set to `home`

        Preconditions (no need to assert):
            home: a valid directory (File object with isDirectory set to True)."""

        # You did not have to assert preconditions, but here is what such an
        # assertion would look like.
        assert isinstance(home,File) and home.isDir, "Bad argument to __init__"
        self.homeDir = home
        self.openDir = home
```

   (b) [11 points] Implement the following method for class Shell.

```
    def addFile(self, n):
        """
        Creates a new File with name set to `n` in the currently open directory.
```

```
        The new File should not be a directory.

        Precondition (no need to assert):
            n: a nonempty string such that no File in the currently open
               directory has name `n`."""
        # STUDENTS: implement this. Hint: make sure you update `contents`

        newFile = File(n, False)
        self.openDir.contents.append(newFile)
```

(c) [16 points] Implement the following method for class Shell.

```
    def cd(self, n):
        """Changes the currently open directory to the one given by name `n` and
        returns None.

        Exceptions:

        If `n` is the empty string,
            * set the open directory to the user's home directory;
            * return None
        If there's no File named `n` in the currently open directory's contents,
            * make no changes;
            * return the string "No such directory".
        If a File with name `n` is in the currently open directory's contents,
        but that file is not a directory,
            * make no changes;
            * return the string "Not a directory".

        Precondition (no need to assert): n is a (possibly empty) string"""

        if n is "":
            self.openDir = self.homeDir
            return
        contents = self.openDir.contents
        for file in contents:
            if file.name == n:
                if file.isDirectory:
                    self.openDir = file
                    return
                else:
                    return "Not a directory"
        return "No such directory"
```

Note: looping through `self.openDir.contents` is necessary. It does not suffice to say something like "`if n not in self.openDir.contents`" because the latter is a list of Files, not strings, and `n` is a string.

---

5. [1 point] **Fill in your Cornell NetID at the top of each page.**

   Also, remember that you should not discuss this exam with students who are scheduled to take a later makeup.

   Always do this! It prevents disaster in cases where a staple fails. And, we need your NetID (a string like LJL2, not your all-numbers Cornell ID number) in order to match up your exam with your Gradescope account.

```
class File:
    """
    Objects represent an instance of a File.
    Some File objects are directories, and some aren't.

    Attributes:
        name [non-empty str]: the name of this File

        isDirectory [boolean]: True if this File is a directory, False otherwise.

        contents [list of Files, or None]:
            * if this File is a directory, `contents` is the list of File objects
              in this directory. `contents` can be empty.
            * if this File is not a directory, then `contents` is None"""

    def __init__(self, n, isDir):
        """Creates a new File with attributes set as follows:
            * name: set to `n`
            * isDirectory: set to `isDir `
            * contents: set to an empty list if `isDir` is True, otherwise None. """
```

Figure 1: Here is the specification for class File and its `__init__` method. You don't have to implement `__init__`, just know from its specification how to create a new File object.