CSCI 134 Practice Final Exam 1 - Solution

This is a 2-hour closed-book examination. All work should be your own. **Simple and concise solutions** will receive the best score.

Keep in mind the following tips:

- You do not need to do the problems in the order they are presented.
- For partial credit on problems, you may explain your strategy with words and pictures.
- If you are taking too long on a problem, move on and come back to it later.
- Your solutions should fit within the provided spaces.

You may use the back of any page as scratch paper. **Good luck!**

Please make sure to include your anonymous ID on **every page** of the exam.

Name: _____Anne Surkey_____

Anonymous ID: _1234____

This exam has 5 questions and 13 total pages. The point breakdown is below.

Question	Points	Score
Question 1	20	
Question 2	5	
Question 3	5	
Question 4	10	
Question 5	20	
TOTAL	60	

Please sign the honor code statement below.

I have neither given nor received unauthorized aid on this exam.

Signed: _____Anne Surkey_____

Question 1: Short Answer (20 points)

Part A. (4 points) Circle all **incorrect** statements regarding dictionaries.

a. Dictionaries are mutable collections of key-value pairs.

b. Values of a dictionary must be unique. **(Keys must be unique in a dictionary; values need not be)**

c. Lists can be used as dictionary keys. **(Dictionary keys must have immutable types; lists are mutable)**

d. Keys of a dictionary must be unique.

Part B. (4 points) Consider the following code:

```
class Course:

    def __init__(self, name, requirements):
        self._name = name
        self._requirements = requirements

    def prereq(self, other):
        """Returns whether `other` is a requirement for this course."""
        return other in self._requirements


    def __str__(self):
        """String representation of object"""
        return self._name


def sort_courses(course_list):
    """Sort the given courses based on prereq using selection sort"""
    for i in range(len(course_list)):
        for j in range(i+1, len(course_list)):
            if course_list[i].prereq(course_list[j]):
                course_list[i], course_list[j] = course_list[j], course_list[i]
    return course_list


if __name__ == "__main__":
    course1 = Course("CS134", set())
    course2 = Course("CS136", {course1})
    course3 = Course("CS256", {course1, course2})
    course4 = Course("ML", {course3})
    order = sort_courses([course3, course4, course2, course1])
    for course in order:
        print(course)
```

What does the above code print to the screen?

Answer:

```
CS134
CS136
CS256
ML
```

Part C. (2 points)
(i) If `course_list` has n elements, then what is the computational complexity of `sort_courses(course_list)`? Express your answer using O-notation.
(ii) Is there a sorting algorithm that can improve the complexity (yes or no)?

Answer:
(i) $O(n^2)$

(ii) Yes

Part D. (4 points) Suppose we wanted to create a subclass of `Course` (defined in Part B) called `IntroductoryCourse`. An `IntroductoryCourse` has no requirements, but otherwise behaves exactly like a regular `Course`. We want to initialize an `IntroductoryCourse` as follows:

```
course = IntroductoryCourse("CS134")
```

To do so, we create the following (incorrect) code:

```
class IntroductoryCourse(Course):
    def __init__(self, name):
        super().__init__(name, set())
```

Fix the above code (there are two errors).

Part E. (3 points) Consider the following function:

```
def mystery(nums):
    if len(nums) == 0:
        return 0
    elif len(nums) == 1:
        return nums[0]
    else:
        return nums[0] * nums[-1] + mystery(nums[1:-1])
```

What is returned by the call `mystery([1,2,3,4,5])`?

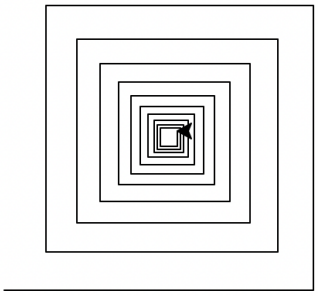
Answer: 16

Part F. (3 points) Consider the following turtle graphics code:

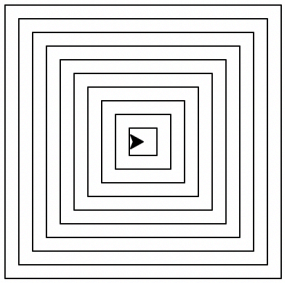
```
def graphics_mystery(length, shrink_factor, angle, min_length):

    if length > min_length:
        fd(length)
        lt(angle)
        graphics_mystery(length*shrink_factor, shrink_factor, angle, min_length)
```

Circle which one of the following `graphics_mystery(200, 0.93, 95, 10)` outputs.

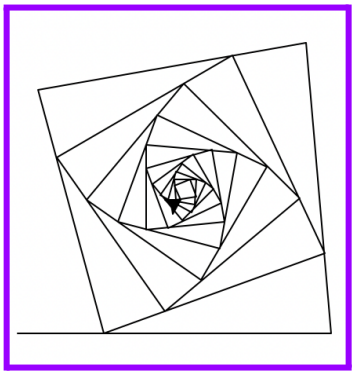
A)



B)



C)



Question 2: Matching (5 points)

Consider a **sorted** list of n numbers which is the input to these algorithms:

A. Algorithm A adds 1 to each item in the list.
B. Algorithm B updates the first item of the list to be zero.
C. Algorithm C compares each number in the list to every other number in the list (using a nested for loop) to determine if the list contains any duplicates.
D. Algorithm D performs binary search on the list.

Fill in the blanks with the letters A through E such that **every statement is correct** and **you don't use the same expression twice**.

Algorithm _____B_____ is a $O(1)$ algorithm.

Algorithm _____D_____ is a $O(\log n)$ algorithm.

Algorithm _____A_____ is a $O(n)$ algorithm.

Algorithm _____C_____ is a $O(n^2)$ algorithm.

Question 3: United Nations (5 points)

Create a function called `countries_by_first_letter` that takes as its only argument a list of strings, each of which is the name of a country. It should return a dictionary that maps each letter of the alphabet to a list consisting of the countries that start with that letter. If there are no countries that start with a particular letter, then that letter should not appear as a key in the returned dictionary.

```
def countries_by_first_letter(countries):
    """Creates a dictionary that maps letters to the countries that begin
    with that letter.

    >>> countries_by_first_letter(['nepal', 'peru', 'norway', 'uganda', 'uruguay'])
    {'n': ['nepal', 'norway'], 'p': ['peru'], 'u': ['uganda', 'uruguay']}

    """
    result_dict = dict()
    for country in countries:
        if len(country) > 0:
            letter = country[0]
            if letter not in result_dict:
                result_dict[letter] = []
            result_dict[letter].append(country)
    return result_dict
```

Question 4: The Life-Changing Magic of Tidying Up, Revisited (10 points)

Part A. (4 points) Palindromes still spark joy, but recursion is suddenly very trendy. Complete the **RECURSIVE** function **`is_palindrome`** that takes a string **`word`** as its argument, and returns a **`boolean`** that indicates whether **`word`** is a palindrome. **Recall:** a palindrome is a word that is spelled the same forward and backward – in each recursive step you should consider shortening the word by excluding the first and last letter of the current word.

```
def is_palindrome(word):
    """Determines if a word is a palindrome.

    >>> is_palindrome("kayak")
    True
    >>> is_palindrome("citric")
    False
    >>> is_palindrome("")
    True
    """
    if len(word) <= 1:
        return True
    else:
        return word[0] == word[-1] and is_palindrome(word[1:-1])
```

Part B. (6 points) Double letters still don't spark joy. Write a **RECURSIVE** function that counts the number of double letters in the string **word**. **Recall:** a double letter is when a letter appears in a string twice in a row. For instance, `"bookkeeper"` has three double letters (`"oo", "kk", "ee"`), whereas `"cuddlepoo"` has two double letters (`"dd", "oo"`). In case you're curious, the word `wheeeeee` has five double letters, since there are five pairs of adjacent `e`'s in the word.

```
def num_double_letters(word):
    """Counts the number of double letters in a word.

    >>> num_double_letters("bookkeeper")
    3
    >>> num_double_letters("cuddlepoo")
    2
    >>> num_double_letters("wheeeeee")
    5
    """
    if len(word) <=1:
        return 0
    elif word[0] == word[1]:
        return 1 + num_double_letters(word[1:])
    else:
        return 0 + num_double_letters(word[1:])
```

Question 5: May Madness! (20 points)

Mark was runner-up in a charity tournament bracket in March, and now thinks of himself as a jock. Let's indulge his delusions. For this question, implement a basketball tournament using Object Oriented Programming.

Your ultimate goal is to create Python classes that allow us to stage a basketball tournament. For instance, we would like to be able to run the following Python code:

```
uconn = Team("uconn", 0.84)
kansas = Team("kansas", 0.86)
florida = Team("florida", 0.81)
amherst = Team("amherst", 0.32)
tourney = Tournament([uconn, kansas, florida, amherst], 50)
tourney.run()
```

and obtain output like the following:

```
uconn vs kansas
...uconn wins!
florida vs amherst
...florida wins!
uconn vs florida
...uconn wins!
```

To do so, you'll need to implement three Python classes: **`Team`**, **`Match`**, and **`Tournament`**.

It will likely be helpful to know the following things about basketball:

- A **`Match`** consists of **`k`** "possessions", and each team gets the same number of possessions.
- During each possession, the team either scores 2 points or 0 points.
- The probability that they score 2 points is determined by the team's **`success_rate`**, a number between 0 and 1 as shown in the creation of the **`Team`** objects above
- If one team has more points than the other after **`k`** possessions, then they are the winner. Else, there is a tie. In the case of a tie, the game goes into overtime, and we'll assume that the team with the higher **`success_rate`** always wins when the game goes to overtime.

Part A. (6 points) Start by filling in the missing code (in the boxes below) in the **`Team` class**. Make sure you:

- Complete the **`__init__`** method, which initializes the **`_name`** and **`_success_rate`** attributes.
- Complete the **`get_name`** method.
- Complete the **`possession`** method.

```
class Team:
```

```
def __init__(self, name, success_rate):
    """Initialize attributes.
    name: a string representing the team's name.
    success_rate: a float between 0 and 1 indicating the percentage of
    possessions during which the team scores a basket.
    """
    self._name = name
    self._success_rate = success_rate
```

```
def get_success_rate(self):
    """Returns the team's success rate."""
    return self._success_rate

def get_name(self):
    """Returns the team's name."""
    return self._name
```

```
def possession(self):
    """Draws a random float between 0 and 1. If that number is less
    than self._success_rate, then we return the integer 2 (since we
    scored a basket). Otherwise, we return the integer 0 (since we
    didn't score a basket).
    """
    from random import random
    rand_float = random()  # chooses a random float between 0 and 1
    if rand_float < self._success_rate:
            return 2
       else:
            return 0
```

Part B: (6 points) Next, implement the **`Match` class**.

```
class Match:

    def __init__(self, team1, team2):
        self._team1 = team1
        self._team2 = team2
```

```
def play(self, k):
    """Plays a basketball match and returns the winning Team.

    Each team gets k possessions, where k is an int. For each possession,
    you should call the .possession method of the team, and then add the
    return value to the team's overall score.

    If one of the two teams has a greater score after k possessions,
    then return that team (the return value of this method should be
    an instance of the Team class).

    If the two teams are tied after all possessions are completed,
    then we enter overtime. In overtime, the team with the higher success rate
    is declared the winner. You can assume two teams never have the same success
    rate (as before, the return value of this method should be
    an instance of the Team class)
    """
    team1_score = 0
    team2_score = 0
    for i in range(k):
        team1_score += self._team1.possession()
        team2_score += self._team2.possession()
    if team1_score > team2_score:
        return self._team1
    elif team2_score > team1_score:
        return self._team2
    elif self._team1.get_success_rate() > self._team2.get_success_rate():
        return self._team1
    else:
        return self._team2
```

Part C: (8 points) Finally, implement the `Tournament` class.

```
class Tournament:
    def __init__(self, teams, num_possessions):
        """teams is a list of instances of the Team class.
        You may assume that teams has at least one element.
        """
        self._teams = teams
        self._k = num_possessions
```

```
def run(self):
    """Simulates a basketball tournament.

    As long as there are at least two teams in self._teams (a
    list of Team instances), the tournament continues. In each
    round of the tournament, the first two teams in self._teams
    play one another. More specifically, the first two teams are
    removed from self._teams and a Match is played between those
    two teams. The winner of the match remains in the tournament,
    and is therefore appended to the end of self._teams.

    This process continues until only one team remains in self._teams.
    This Team should be the return value of this method.

    For each round, you should also print the teams in the
    current Match and the winner of the Match.
    """
```

```
    while len(self._teams) > 1:
        next_match = Match(self._teams[0], self._teams[1])
        winner = next_match.play(self._k)
        print(self._teams[0].get_name() + " vs " + self._teams[1].get_name())
        print("..." + winner.get_name() + " wins!")
        self._teams = self._teams[2:] + [winner]
       return self._teams[0]
```

Reference Sheet

You are free to use any definitions from homeworks, labs, or lectures. Here is a non-exhaustive summary of useful functions, types, and methods.

`range(start, stop, step)`:
`range` object generating integers starting at `start` (by default 0), going up to `stop` in steps `step`

`randint(start, end)`: (from the `random` module)
returns an integer `i` such that `start <= i <= end` (inclusive of both `start` and `end`)

`list` class:

```
L.append(object) -> None
  appends object to end of L
L.count(item) -> int
  returns number of times item appears in
L.extend(iterable) -> None
  extends L by appending elements from the iterable
L.index(value) -> int
  returns the first index of value (or error, if not found).
L.insert(index, object)
  inserts object before index in L
L.remove(item) -> None
 removes the first instance of item from L
L.sort() -> None -- Sorts list L
```

`tuple class:`

```
T.count(val) -> int
  Returns number of occurrences of val.
T.index(val) -> int
  Returns the first index of value.
```

`dict class:`

```
D.get(key, default=None) -> object
    Returns the value for key if key is in the dictionary, else default.
D.keys() -> object
    Returns a set-like object providing a view of D's keys
D.items() -> object
    Returns an object providing a view of D's items as a list of (key, value) tuple pairs
```