15-112 Spring 2019
Practice Final
180 minutes

Name:

Andrew ID:

Recitation Section:

- **This practice final is written by TAs and may not reflect the true difficulty or length of the actual final.**
- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- You may use the backs of pages as scratch paper. Nothing written on the back of any pages will be graded.
- You may complete the problems in any order you'd like; you may wish to start with the free response problems, which are worth most of the credit.
- All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.
- Do not use these concepts: generators or anything not taught in 112

Don't write anything in the table below.

Question	Points	Score
1	10	
2	10	
3	5	
4	5	
5	7	
6	13	
7	10	
8	15	
9	15	
10	10	
Total:	100	

1. **Code Tracing**

   Indicate what each piece of code will print. Place your answer (and nothing else) in the box below each piece of code.

   (a) (3 points) CT1

```
def ct1(x, d, *args):
    print(" " * d + str(args) + " " + str(x))
    if len(args) <= 2:
        result = x
    else:
        mid = len(args) // 2
        res1 = ct1(x + len(args), d+1, *args[:mid])
        res2 = ct1(x + sum(args), d+1, *args[mid:])
        result = res1 + res2
    print(" "*d + "--> " + str(result))
    return result

print(ct1(0, 0, 7, 3, 5, 1, 2))
```

(b) (3 points) CT2

```
import copy
def ct2(a):
    b = a
    c = copy.copy(a)
    a = a + [a[0]]
    d = copy.deepcopy(c)

    a[0][0] += 5
    c[1].extend(['term', 'project'])
    b.append(d.pop(0))
    b[3] += [112]

    b = b[:]
    a[1] = [2]
    c[1][1] = 42

    print('a', a)
    print('b', b)
    print('c', c)
    print('d', d)


a = [[1, 5], [2], 'yeet']
ct2(a)
print(a)
```

(c) (4 points) CT3

```
import functools
def ct3(L, x):
    def f(*args):
        return x // (len(list((filter(lambda y: y < 2, args)))) + len(args))
    print(L, x)
    if f(*L) < 2:
        return L
    else:
        L = list(map(lambda y: y // 2, L))
        x = functools.reduce(lambda a,b: a*b, L, 1)
        return ct3(L, x)


print(ct3([5, 7, 6, 4, 3], 42))
```

2. **Reasoning Over Code**

For each function, find parameter values that will make the function return True. Place your answer (and nothing else) in the box below each block of code.

(a) (3 points) ROC1

```
def roc1(n):
    i = 0
    p = 0
    while (n > 0):
        m = n % 100
        assert(m > p)
        (i, n, p) = (i+1, n//100, m)
    return ((i == 4) and (i + p == 100))
```

(b) (3 points) ROC2

```
def roc2(s):
    def f(s):
          if (len(s) < 2): return s
          else: return s[-1] + f(s[:-1])
    t = ""
    while (s != ""):
          t += s[0]
          s = f(s[1:])
    return (t == "Yes!")
```

(c) (4 points) ROC3

```
def roc3(L):
    assert(L == list(reversed(sorted(L))))
    assert(L != list(reversed(sorted(set(L)))))
    assert(len(L)  %  2 == 0)
    assert(max(L)  <  10)
    def f(L):
        if  (L  ==  [   ]): return  [   ]
        else:   return  f(L[1:-1]) + [L[0]  *  L[-1]]
    return  (f(L)   ==  [20,10])
```

3. **Short Answer**

Answer each of the following *very briefly*.

(a) (1 point) Briefly describe how hashing makes it possible to perform constant-time lookups in a set.

(b) (1 point) In class we studied P vs NP and we also studied the game of Sudoku. Assume that you have just written code that solves a Sudoku puzzle in O(N^{28}) time where the Sudoku board has dimensions N x N. Does this mean P = NP? (True or False) Briefly explain your answer.

(c) (1 point) How does memoization improve the efficiency of recursive code?

(d) (1 point) Briefly describe the law of large numbers.

(e) (1 point) True or False. We can write a function called `halts(f, input)` that returns True if the function `f` halts/terminates on the provided `input` and returns False otherwise. Briefly explain your answer.

4. **Big-Oh**

For each function write its most simplified Big-Oh runtime in terms of N.

(a) (3 points) Big-Oh 1

```
def bigOh1(L):
    # Assume N = len(L)
    result = [0 for i in range(len(L)**2)]
    i = 1
    while(i < len(L)):
        result[i] = i**2
        i *= 2
        result = result + L
    return result
```

(b) (2 points) Big-Oh 2

```
from string import ascii_lowercase

def bigOh2(s):
    # Assume N = len(s)
    res = {}
    for c in ascii_lowercase:
        if c in s:
            res[c] = s.count(c)
        else:
            print(s)
    return
```

5. (7 points) **Free Response: nthConceitedNumber**

   Write the function `nthConceitedNumber(n)`, where a conceited number is a k-digit number that is the sum of the k'th powers of all of its digits.

   For example, 153 is conceited because $153 == 1^3 + 5^3 + 3^3$

   The first few conceited numbers are 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, and 371

   For any credit, **you may not use strings, lists, or dictionaries in your solution.**

6. (13 points) **Free Response: spiralJoin**

Write the function `spiralJoin(L)` that takes in a rectangular or square 2d list L of strings, and returns a string of the elements in the matrix concatenated in spiral order. (You can assume L is rectangular/square and not ragged) Here is an example to clarify what spiral order means.

Let us assume that:

L =

[”2”, ”3”, ”4”],

[”5”, ”6”, ”7”],

[”8”, ”ef”, ”10”]]

In this case we should return ”234710ef856”. As you can see, we go along the top row of L first, adding ”2”, ”3” and ”4” to our result string. Then we go down the right edge, and then left across the bottom row, and then up the left edge. We repeat this pattern until we have added all elements from our matrix.

Hint: There are many ways to approach this problem, but one possible way would be to traverse the matrix in this spiral order, changing directions as necessary. Notice that we start off traversing right, then when we cannot traverse right any further we start traversing down. Once we cant traverse down any further we traverse left and so on. Think carefully about in which cases you need to switch the traversal direction and how you can keep track of the current direction you are heading in.

Here is a test function to further help clarify the problem.

```
def testSpiralJoin():
    print("Testing spiralJoin")
    L = [["a", "b", "c", "de"],
         ["fgh", "ijk", "lm", "n"],
         ["q", "rse", "t", "u"]]
    L1 = [["18", "3"],
         ["gg", "qq"]]
    L2 = [["eat"]]
    assert(spiralJoin(L) == "abcdenutrseqfghijklm")
    assert(spiralJoin(L1) == "183qqgg")
    assert(spiralJoin(L2) == "eat")
    print("testing passed")
```

Additional Space for Answer to Question 6

7. (10 points) **Free Response: numberWooTriple(L)**

   Write the function `numberWooTriple(L)` that takes a list of positive, integers and returns a set of Woo Triples as tuples that can be created using the elements in the list. A Woo triple is a triple of numbers where $a^2 \% b = c$.

   For example, `numberWooTriple([1,3,2])` returns {`(1, 1, 3), (1, 2, 3), (1, 1, 2)`}, because (1, 1, 3), (1, 2, 3), and (1, 1, 2) are Woo triples as:

   $1^2 \% 3 == 1$

   $2^2 \% 3 == 1$

   $1^2 \% 2 == 1$

   All triples can be made with the elements in the given list. Note that it is ok to duplicate an element, as (1, 1, 3) has use the number 1 twice.

   Also note that (1, 2, 3) and (3, 2, 1) are the same triple, and your function should only return one of these triples.

   Note: For any credit your solution must run in O(N^2) time, where N is the length of the input list L.

Additional Space for Answer to Question 7

8. (15 points) **Free Response: getSquarefulArrangement**

   Write the function `getSquarefulArrangement(L)`, that given a 1d list of integers `L` returns a permutation of the list that is squareful if such a solution exists and returns None otherwise.

   A list L is said to be *squareful* if every pair of adjacent numbers in the list sum to a perfect square.

   Here is an example to clarify the problem. Given:

   ```
   L = [1, 17, 8]
   ```

   one possible permutation that the function could return is [1, 8, 17]. Notice that [1, 8, 17] is squareful as $1 + 8 = 9$ is a perfect square and $8 + 17 = 25$ is also a perfect square. On the other hand [17, 1, 8] would not be a valid solution as $17 + 1 = 18$ is not a perfect square.

   To receive credit for this solution you must use recursive backtracking. Furthermore, you must build up the solution as you go. A solution that simply generates all the permutations of the input list and then checks which permutations are squareful will receive no credit

Additional Space for Answer to Question 8

9. (15 points) **Free Response: Circles and Bullseye Classes**

This is a 2 part problem. First write the necessary classes so that the following test cases all pass.

```
# first parameter is x co-ordinate of circle center
# second parameter is y co-ordinate of circle center
# third parameter is radius of circle
c1 = Circle(50, 90, 10)
assert(c1.x == 50)
assert(c1.y == 90)
assert(c1.r == 10)
assert(Circle.numCircles == 1)
c2 = Circle(50, 100, 10)
assert(Circle.numCircles == 2)
# circles are the same if they have the same radius
assert(c1 == c2)
c3 = Circle(50, 100, 8)
assert(c1 != c3)
assert(c2 != c3)
assert(c2 != "Do not crash!")
assert(str(c1) == "Circle at (50, 90) with radius 10")
# move() moves a circle down in the y direction by 10 pixels
c1.move()
assert(c1.x == 50)
assert(c1.y == 100)

# bullseyes look like a ring with numRings number of smaller rings inside
# each ring has a radius that is 2 pixels smaller than the outer ring
# for example, this bullseye has a radius of 12
# the ring inside it will have a radius of 10
# and the ring inside that one will have a radius of 8
# first parameter is x co-ordinate of bullseye center
# second parameter is y co-ordinate of bullseye center
# third parameter is radius of bullseye
# fourth parameter is the number of rings in the bullseye
s1 = Bullseye(100, 200, 12, 3)
assert(s1.x == 100)
assert(s1.y == 200)
assert(s1.r == 12)
assert(s1.numRings == 3)
s2 = Bullseye(95, 250, 12, 3)
s3 = Bullseye(80, 250, 11, 7)
# bullseyes are the same if their radius and number of rings are the same
assert(s1 == s2)
assert(s1 != s3)
```

```
assert(str(s1) == "Bullseye at (100, 200) with radius 10 and 3 rings")
# bullseye moves in the same way as a circle
s1.move()
assert(s1.x == 100)
assert(s1.y == 210)
# we can add both circles and bullseyes to sets
cset = set()
cset.add(c1)
assert(c1 in cset)
assert(c2 in cset)
assert(c3 not in set)
cset.add(s1)
assert(s1 in cset)
assert(s2 in cset)
assert(s3 not in cset)
```

Next, using the classes you have written, assuming the run function has been written for you, finish up your animation code with the following specifications:

- Add in a draw method to your classes that draws the circles and bullseyes as required
- If you press the 'c' key, a new circle is added on the screen with a random location on the screen and random radius between 10 and 30.
- If you click on the screen, a bullseye with a random number of rings between 2 and 5 and a random radius between 15 and 30 is added centered where the mouse was clicked.
- You must use a set to keep track of all circles and bullseyes within data.
- You should only generate unique circles and bullseyes. Remember that 2 circles are considered the same if they have the same radius, and 2 bullseyes are considered the same if they have the same radius and the same number of rings. Note: do not worry about what will happen if your code runs forever.
- Every 5 seconds, all circles should be removed from the screen. The bullseyes should not be removed, only the circles.

Additional Space for Answer to Question 9

Additional Space for Answer to Question 9

10. (10 points) **Free Response: bestNameFile**

Write the function `bestNameFile(path, name)` that given a `path` to a folder and a string describing a `name` to search for, returns the path to the file with the maximum number of occurrences of the `name` in its contents. You must consider all files inside the provided folder and any of its subfolders recursively.

Note that we do not care about casing. For example if `name` = "raunak", then both "RaUNak" and "RAUNAK" count as an occurence.

You may assume that you have access to the function `readFile(path)` that when called on a path that represents a file, returns a string of all the contents of that file.

You may use iteration for this problem as long as you use recursion in a meaningful way.

Additional Space for Answer to Question 10