15-112
Spring 2019 Exam 2
April 4, 2019

Name:

Andrew ID:

- 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.
- All code samples run without crashing. Assume any imports are already included as required.
- You may assume that math, string, tkinter, and copy are imported; do not import any other modules.

Don't write anything in the table below.

Question	Points	Score
1	15	
2	10	
3	10	
4	10	
5	15	
6	20	
7	20	
Total:	100	

1. **Short Answer**

Answer each of the following *very briefly*.

(a) (4 points) Consider the following four classes that have been implemented by the designers of a game: Monster, Player, Bat, FireBat. Assuming inheritance is properly used, fill in the following blanks. If the answer to a blank is "nothing", then write "nothing".

Monster is a subclass of _____.

Monster is the superclass of _____.

Player is a subclass of _____.

Player is the superclass of _____.

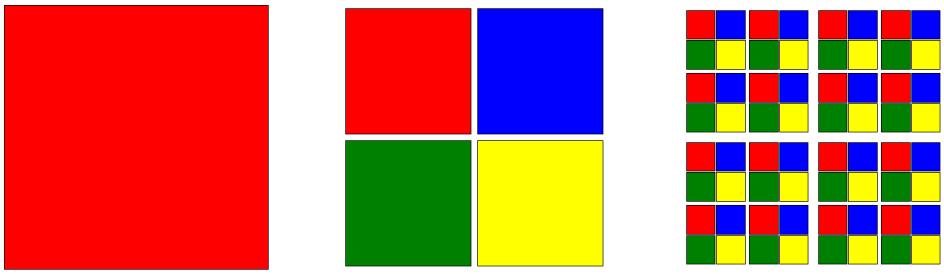
Bat is a subclass of _____.

Bat is the superclass of _____.

FireBat is a subclass of _____.

FireBat is the superclass of _____.

(b) (4 points) Consider the following fractal drawn at levels 0, 1, and 3.



Assume that the function `drawSimonFractal` is called to draw this fractal. In the code, how many calls to `drawSimonFractal` and `canvas.create_rectangle` occur in the base case? What about the recursive case?

Base Case		**Recursive Case**	
Number of calls to `drawSimonFractal`		Number of calls to `drawSimonFractal`	
Number of calls to `canvas.create_rectangle`		Number of calls to `canvas.create_rectangle`	

(c) (2 points) A filesystem is a naturally recursive data structure, and therefore has a base case and a recursive case in the data representation. Using **only one word** in each answer, what is a filesystem's:

Base case? _____

Recursive case? _____

(d) (2 points) You may recall from the notes that selection sort is $O(n^2)$ and merge sort is $O(n\ log\ n)$. In certain situations, however, a good implementation of selection sort can be faster than a good implementation of merge sort. How can this be?

(e) (3 points) What is the difference between a class and an instance? Write a small amount of code that creates a class and code that creates an instance.

2. **Code Tracing**

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

(a) (5 points) CT1

```
def ct1(s, d=0):
    print(d, "in: ", s)
    if len(s) == 0:
        res = ""
    else:
        res = s[::2]+ct1(s[1::2], d+1)
    print(d, "out: ", res)
    return res

ct1("abcdefgh")
```

(b) (5 points) CT2

```
def ct2(L):
    d = dict()
    for i in range(len(L)):
        d[sum(L[i:])] = L[i]
    print("1:",d)

    for item in L:
        if item in d:
            d[item] += 1
    print("2:",d)

    b = copy.deepcopy(d)
    for k in b:
        if d[k] % 2 == 1:
            del d[k]
    return d

print("3:",ct2([1,2,3,4]))
```

3. **Reasoning Over Code**

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

(a) (5 points) ROC1

```
def roc1(lst):
    assert(len(lst) == 8 and isinstance(lst, list))

    for item in lst:
        if not isinstance(item, int) and not isinstance(item, str):
            return False

    s = set()
    for item in lst:
        if type(item) == int:
            s.add(lst[item])
        elif type(item) == str and len(item) != 1:
            return False

    return "".join(sorted(list(s))) == "act"
```

(b) (5 points) ROC2

```
def f(s):
    if (len(s) < 2):
            return s
    else:
        return s[-1] + f(s[:-1])

def roc2(s):
    t = ""
    while (s != ""):
        t += s[0]
        s = f(s[1:])
    return (t == "Nice")
```

4. (10 points) **Big-O:** For each function shown below, write next to each line of the function either the Big-O runtime of the line or the number of times the line loops. Then write the total Big-O runtime of the function in terms of N in the box to the right of the code. **All answers must be simplified- do not include lower-order terms! For full credit, you must include line-by-line Big-O.**

```
def f1(s): # s contains N characters          # Big-O
    i = 1                                     #______
    sum = 0                                   #______
    while i < len(s):                         #______
        sum += (ord(s[i]) - ord('a'))         #______
        i *= 2                                #______
    c = chr((sum % 26)+ord('a'))              #______
    return c                                  #______
```

```
def f2(L): # L contains N items               # Big-O
    allowedWords = ["cat", "dog", "rat"]      #______
    newList = []                              #______
    for w in L:                               #______
        if w in allowedWords:                 #______
            newList.append(w)                 #______
    for w in allowedWords:                    #______
        if w in L:                            #______
            L.remove(w)                       #______
        L.sort()                              #______
    L.extend(newList)                         #______
    return L                                  #______
```

```
def f3(L): # L contains N items               # Big-O
    s = set()                                 #______
    for i in L:                               #______
        for j in range(len(L)):               #______
            s.add(i+j)                        #______
    lst = list(s)                             #______
    lst.sort()                                #______
    lst.reverse()                             #______
    return lst                                #______
```

5. (15 points) **Free Response: Recursive getHiLo**

   Write the recursive function `getHiLo(lst)` which, given a list of integers `lst` returns a tuple contain the highest and lowest values in the list. For example:

   `getHiLo([1, 7, 3, 8, 2, 9, 6, 4, 5])` returns `(9,1)`

   `getHiLo([5])` returns `(5,5)`

   `getHiLo([])` returns `None`

   Your solution must use recursion. If you use any loops, comprehensions, or iterative functions, you will receive no points on this problem.

6. (20 points) **Free Response: Stapler and ElectricStapler**

Write the classes Stapler and ElectricStapler so that the following test code runs without errors. Do not hardcode against the values used in the testcases, though you can assume the testcases cover the needed functionality. You must use proper object-oriented design, including good inheritance, or you will lose points

```
# A stapler starts out full of the specified number of staples
s = Stapler(50)
assert(str(s) == "Stapler(50/50)")
# You can staple something if the stapler has staples
s.staple()
assert(str(s) == "Stapler(49/50)")
# You can refill the stapler to get it back to full.  Full is
# the number of staples it started with, which could be different
# from stapler to stapler.
s.refill()
assert(str(s) == "Stapler(50/50)")
for i in range(50):
    s.staple()
assert(str(s) == "Stapler(0/50)")
# If you try to staple while empty, nothing changes
s.staple()
assert(str(s) == "Stapler(0/50)")

# An electric stapler always starts with 500 staples, and is unplugged
e = ElectricStapler()
assert(str(e) == "DeadStapler(500/500)")
# You can plugin an electric stapler
e.plugIn()
assert(str(e) == "ElectricStapler(500/500)")
# An electric stapler can also staple and refill
e.staple()
assert(str(e) == "ElectricStapler(499/500)")
e.refill()
assert(str(e) == "ElectricStapler(500/500)")
# But an unplugged electric stapler can't staple, you get a "No Power" exception
e.unPlug()
ok = False
try:
    e.staple()
except:
    ok = True
assert(ok)
# Like a normal stapler, an electric stapler also doesn't change if you
# staple while empty
e.plugIn()
for i in range(500):
    e.staple()
assert(str(e) == "ElectricStapler(0/500)")
e.staple()
assert(str(e) == "ElectricStapler(0/500)")

# Checking inheritance
assert(isinstance(s, Stapler) == True)
assert(isinstance(s, ElectricStapler) == False)
assert(isinstance(e, Stapler) == True)
assert(isinstance(e, ElectricStapler) == True)
```

Additional Space for Answer to Question 6

Additional Space for Answer to Question 6

7. (20 points) **Free Response: findRTP(digits)**

   Background: A number n is a right-truncatable prime, or RTP, if every prefix of n (including n itself) are all prime. So, 593 is an RTP because 5, 59, and 593 are all prime.

   With this in mind, write the function `findRTP(digits)` that takes a positive int, `digits`, and returns the smallest RTP with that many digits, or `None` if no such number exists.

   To do this, you must use backtracking. At each step, try to add one more digit to the right of the number. Also, make sure you are only creating RTPs as you go.

   You may assume that `isPrime(n)` is already written for you.

   While not required, you may find it is helpful to use an optional parameter, `prefix`, which is 0 by default and which holds the number that has been constructed so far.

   Note: Even though `findRTP(8)` returns `23399339`, it runs almost instantly because backtracking rules out most numbers without trying them, so it actually calls `isPrime` very few times.

Additional Space for Answer to Question 7

15-112
Spring 2019 Exam 2 Handout
April 4, 2019

Name:

Andrew ID:

- This handout contains reference material from the exam.
- **You must turn this in with your exam, but nothing on this reference will be graded**.

General	
Function/Method	**Complexity**
Print	O(N)
Range in Iteration	Number of iterations = (end - start)/step

Strings: s is a string with N characters	
Function/Method	**Complexity**
Len	O(1)
Membership	O(N)
Get single character	O(1)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
Chr() and Ord()	O(1)
Concatentation	O(len(s1) + len(s2))
Character Type Methods	O(N)
String Edit Methods	O(N)
Substring Search Methods	O(N)

Lists: L is a list with N elements	
Function/Method	**Complexity**
Len	O(1)
Append	O(1)
Extend	O(K)
Concatentation with +=	O(K)
Concatentation with +	O(N + K)
Membership Check	O(N)
Pop Last Value	O(1)
Pop Intermediate Value	O(N)
Count values in list	O(N)
Insert	O(N)
Get value	O(1)
Set value	O(1)
Remove	O(N)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
Sort	O(N log (N))
Multiply	O(N*D)
Minimum	O(N)
Maximum	O(N)
Copy	O(N)
Deep Copy	O(N*M)

Sets: s is a set with N elements	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Adding an Element	O(1)
Removing an Element	O(1)
Union	O(len(s) + len(t))
Intersection	O(min(len(s), len(t)))
Difference	O(len(s))
Clear	O(len(s))
Copy	O(len(s))

Dictionaries: d is a dictionary with N key-value pairs	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Get Item	O(1)
Set Item	O(1)
Delete Item	O(1)
Clear	O(N)
Copy	O(N)

```
# A stapler starts out full of the specified number of staples
s = Stapler(50)
assert(str(s) == "Stapler(50/50)")
# You can staple something if the stapler has staples
s.staple()
assert(str(s) == "Stapler(49/50)")
# You can refill the stapler to get it back to full.  Full is
# the number of staples it started with, which could be different
# from stapler to stapler.
s.refill()
assert(str(s) == "Stapler(50/50)")
for i in range(50):
    s.staple()
assert(str(s) == "Stapler(0/50)")
# If you try to staple while empty, nothing changes
s.staple()
assert(str(s) == "Stapler(0/50)")

# An electric stapler always starts with 500 staples, and is unplugged
e = ElectricStapler()
assert(str(e) == "DeadStapler(500/500)")
# You can plugin an electric stapler
e.plugIn()
assert(str(e) == "ElectricStapler(500/500)")
# An electric stapler can also staple and refill
e.staple()
assert(str(e) == "ElectricStapler(499/500)")
e.refill()
assert(str(e) == "ElectricStapler(500/500)")
# But an unplugged electric stapler can't staple, you get a "No Power" exception
e.unPlug()
ok = False
try:
    e.staple()
except:
    ok = True
assert(ok)
# Like a normal stapler, an electric stapler also doesn't change if you
# staple while empty
e.plugIn()
for i in range(500):
    e.staple()
assert(str(e) == "ElectricStapler(0/500)")
e.staple()
assert(str(e) == "ElectricStapler(0/500)")

# Checking inheritance
assert(isinstance(s, Stapler) == True)
assert(isinstance(s, ElectricStapler) == False)
assert(isinstance(e, Stapler) == True)
assert(isinstance(e, ElectricStapler) == True)
```