YOUR NAME PLEASE:

NETID:

Computer Science 200b
Exam 1 - Practice
March 2016

Enter your netid at the bottom of each page.

Closed book and closed notes. No electronic devices. Show ALL work you want graded on the test itself. You may not hand in a Blue Book.

For problems that do not ask you to justify the answer, an answer alone is sufficient. However, if the answer is wrong and no derivation or supporting reasoning is given, there will be no partial credit.

GOOD LUCK!

Problem	Points	Actual
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
Total	60	

1.(a) (5 points)

Define a Python procedure roundup(base,lst) that changes every top-level item of lst that is not an even multiple of base to the next highest integer which is an even multiple. You may assume the lst contains only integers. Do not write any auxiliary procedures for this problem. You may use iteration or recursion.

Examples:

```
>>> roundup(2,[1,2,3,4,5,6]) => [2, 2, 4, 4, 6, 6]
>>> roundup(2,[2,4,6,8]) => [2, 4, 6, 8]
>>> roundup(3,[1,2,3,4,5,6]) => [3, 3, 3, 6, 6, 6]
>>> roundup(4,[1,2,3,4,5,6]) => [4, 4, 4, 4, 8, 8]
>>> roundup(4,[-1,-2,-3,-4,-5,-6]) => [0, 0, 0, -4, -4, -4]
```

```
def roundup(base, l):
    lst = l[:]
    for i,n in enumerate(lst):
        r = n % base
        if r != 0:
            lst[i] = n + (base - r)
    return lst
```

1.(b) (5 points)

Define the same procedure above, roundup(base, lst), as a list comprehension.

```
def lcroundup(base, lst):
    return [n + (base - (n % base)) if n % base != 0 else n for n in lst]
```

2. (10 points)

Define a class employee and associated methods that has the following behavior.

```
e1 = employee("John", 30000)
e2 = employee("Mary", 40000)
e3 = employee("Jane", 50000)
e4 = employee("Hannah", 60000)

e1.add_supervisor(e3)
e2.add_supervisor(e3)
e3.add_supervisor(e4)

employee.members => [employee('John', 30000), employee('Mary',
40000), employee('Jane', 50000), employee('Hannah', 60000)]

employee.highest_paid() => ('Hannah', 60000)

e4.all_reports() =>
employee('Jane', 50000)
employee('John', 30000)
employee('Mary', 40000)
```

```
class employee:
    members = []

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        employee.members.append(self)
        self.supervisor = ''
        self.reports = []

    def __repr__(self):
      return "employee({}, {})".format(repr(self.name), repr(self.salary))

    def add_supervisor(self, item):
        # should check to see if already has a supervisor
        self.supervisor = item
        item.reports.append(self)


    def all_reports(self):
        # should check for cycles
        for r in self.reports:
            print (r)
```

```
        r.all_reports()

@staticmethod
def highest_paid():
  max = 0
  maxname = ''
  for e in employee.members:
        if e.salary > max:
        max = e.salary
        maxname = e.name
  return (maxname, max)
```

3. (10 points) Define a python procedure changetree(new, old, tree) that replaces every occurrence of old in tree with new.

Do not write any auxiliary procedures for this problem.

Examples:

```
changetree(1,2,[1,2,3,1,4,5]) => [1, 1, 3, 1, 4, 5]
changetree(1,2,[1,2,[1,2]]) => [1, 1, [1, 1]]
changetree(1,2,[2,4,[2,4,[[[2,4]]]]]) => [1, 4, [1, 4, [[[1, 4]]]]]
changetree('tulip','rose',['a','rose',['is','a',['rose']],['is','a','rose']])
=> ['a', 'tulip', ['is', 'a', ['tulip']], ['is', 'a', 'tulip']]
```

```
def changetree(new, old, tree):
    if not tree:
        return tree
    if tree == old:
        return new
    if type(tree) is not list:
        return tree
    result = []
    for i in tree:
        result.append(changetree(new, old, i))
    return result
```

4. (10 points)

Regular expressions. For each pattern, list the strings that will match.

Patterns:

```
   A. 'aaa'            1
   B. '...'            1 2 3 4 5 6 7 8 9
   C. '^...$'          1 2 3 4 9
   D. '\.\.\.'         5 6
   E. '^[aeiou]*$'     1
   F. '^[^aeiou]*$'    2 4 5 9
   G. '\w\W\w'         7 8
   H. '^\d+$'          4
   I. '^[0-7]+$'       4
   J. '^[0-9A-Fa-f]+$' 1 2 3 4 9
   K. '^[a-z]$'        (none)
   L. '^\s+'           8
   M. '^\d\s?'         4 5
```

```
Strings:
   1. 'aaa'
   2. 'bbb'
   3. 'abc'
   4. '123'
   5. '123...456'
   6. 'abc...def'
   7. 'a.b.c.d'
   8. ' a b c '
   9. 'AAA'
```

see http://zoo.cs.yale.edu/classes/cs200/lectures/f0217.py

5. (10 points)

Write the UNIX command(s) corresponding to **XXXX** in the transcript below.

```
-bash-4.2$ pwd
/home/accts/sbs5/cs201
-bash-4.2$ ls
bin  class  graded  handouts  hws  previous-years  README  SUBMIT
TESTS  www
-bash-4.2$ mkdir mt
-bash-4.2$ cd mt
-bash-4.2$ pwd
/home/accts/sbs5/cs201/mt
-bash-4.2$ ls .. > f1
-bash-4.2$ ls -l
total 4
-rw-rw-r-- 1 sbs5 cs201ta 72 Oct  6 16:12 f1
-bash-4.2$ cat f1
bin
class
graded
handouts
hws
mt
previous-years
README
SUBMIT
TESTS
www
-bash-4.2$ cp f1 f2
-bash-4.2$ ls -l
total 8
-rw-rw-r-- 1 sbs5 cs201ta 72 Oct  6 16:12 f1
-rw-rw-r-- 1 sbs5 cs201ta 72 Oct  6 16:13 f2
-bash-4.2$ diff f1 f2
-bash-4.2$ rm f1
-bash-4.2$ ls -l
total 4
-rw-rw-r-- 1 sbs5 cs201ta 72 Oct  6 16:13 f2
-bash-4.2$ wc f2
11 11 72 f2
```

6. (10 points) Evaluate the following Python expressions.

(Remember to show work to permit partial credit!)

```
Example: 'hello'[-1] => 'o'
```

```
(a) '&'.join(str(1234))  => '1&2&3&4'
```

```
(b) list(map(lambda x: x == x[::-1], ['bob', 'john', 'sue'])) =>
[True, False, False]
```

```
(c) sorted(['ted', 'donald', 'marco', 'john'], key=len, reverse=True) =>
['donald', 'marco', 'john', 'ted']
```

```
(d)  {i: i*i for i in range(5)} => {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

```
(e) (lambda x: x * 3 + 1 if x % 2 else x / 2)(11) => 34
```