

15-112 F24
Midterm1 version B
(80 min)

Name: _____

Andrew ID: _____@andrew.cmu.edu

Section: _____

- You may not use any books, notes, or electronic devices during this exam.
- Exam versions are color-coded. You must have a different version (color) of this exam than the students sitting to your left and right.
- Stop writing and submit the entire quiz when instructed by the proctor.
- Do not unstaple any pages.
- You must submit the entire quiz with all pages intact.
- Discussing the exam with anyone in any way, even briefly, is cheating. (You may discuss it only once the exam has been posted to the course website.)
- You may not use your own scrap paper. If you must use additional scrap paper, raise your hand and we will provide some. You must hand any scrap paper in with your paper exam, and we will not grade it.
- You may not ask questions about the exam except for English-language clarifications. If you are unsure how to interpret a problem, take your best guess.
- All code samples run without crashing. Assume any imports are already included as required.
- You may not use any concepts (including builtin functions) which we have not covered in the notes in weeks 1-5 / units 1-5.
- You may not use dictionaries, sets, recursion, or OOP.
- We may test your code using additional test cases.
- Assume `almostEqual(x, y)` and `rounded(n)` are both supplied for you. You must write all other helper functions you wish to use, unless we specify otherwise.

Code Tracing (CT) [25 pts, 2.5 pts each]

For each CT, indicate what the code prints

Place your answer (and nothing else) in the box below the code.

CT1:

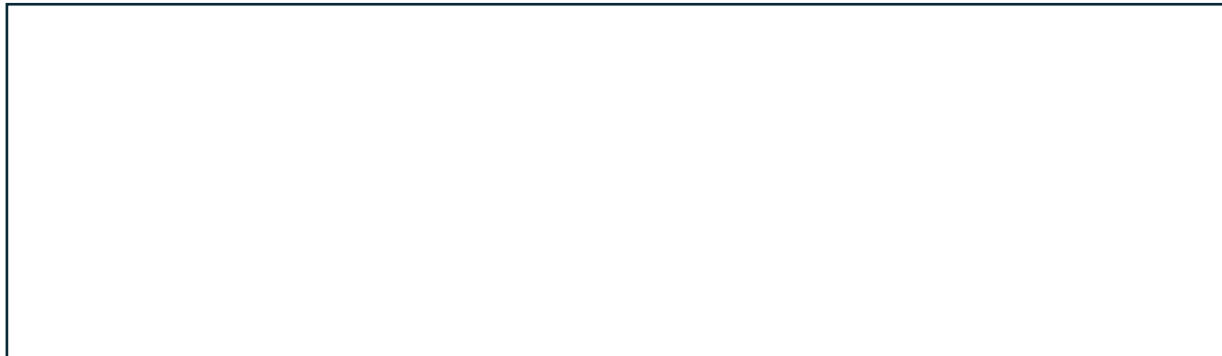
```
def ct1(x, y):  
    L = [ [ 2 ] ]  
    print(int(type(4) == type(4/2)))  
    print(int(type(L) == type(L[0])))  
    return (x%y, y%x, x//y, y//x)  
print(ct1(18, 5))
```

CT2:

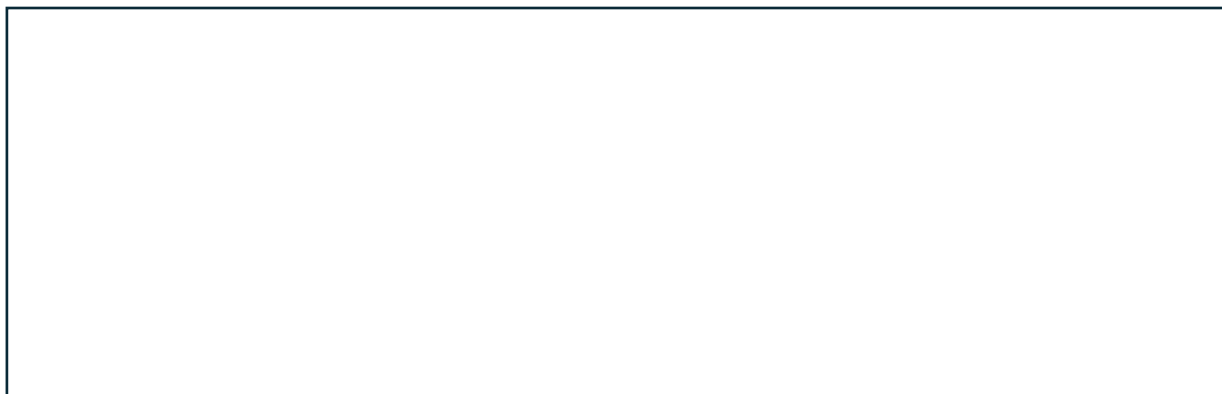
```
def ct2(x, y):  
    print(x*y, x**y, 3*x+y%2)  
print(ct2(2, 3))
```

CT3:

```
def f(x): return 2*x
def g(x): return f(2*f(x))
def ct3(x):
    return x + f(g(x))
print(ct3(4))
```

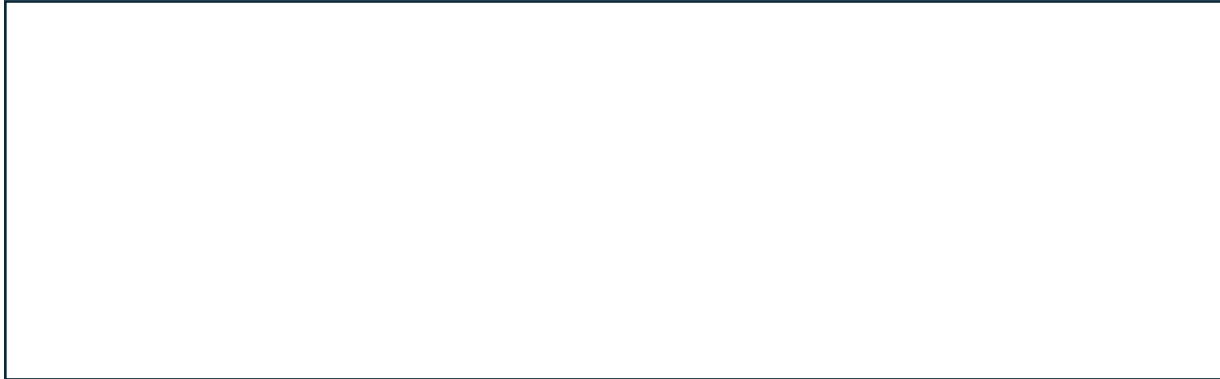
**CT4:**

```
def ct4(n):
    for x in range(n, 3*n, 3):
        if x % 10 == 0:
            continue
        print(x, end='x')
        if x % 10 == 6:
            break
    return x
print(ct4(7))
```

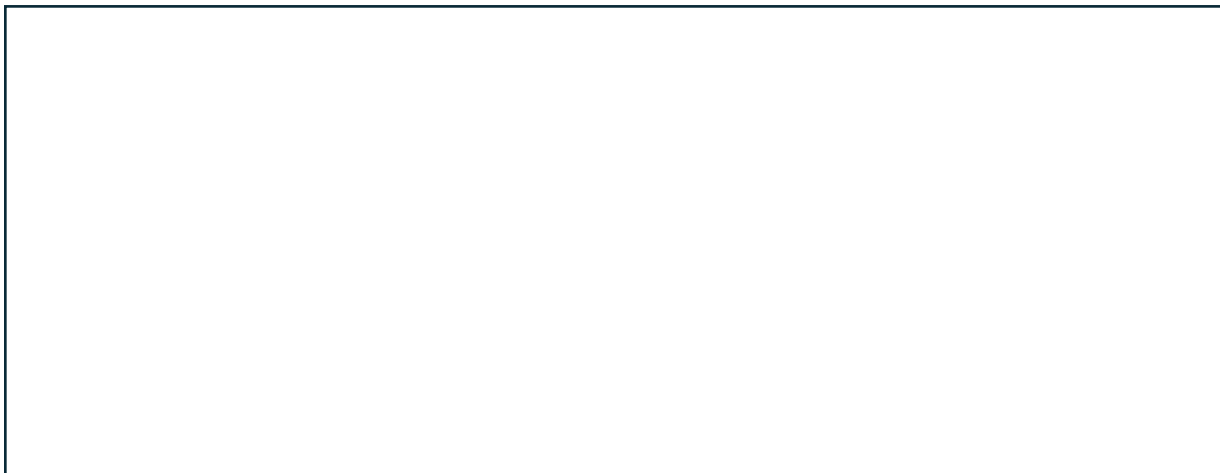


CT5:

```
def ct5(s):  
    t = s.upper().replace('CD', 'F').replace('FF', 'car')  
    t = t[:1] + t[2:]  
    return t[::-1]  
print(ct5('cdCDe'))
```

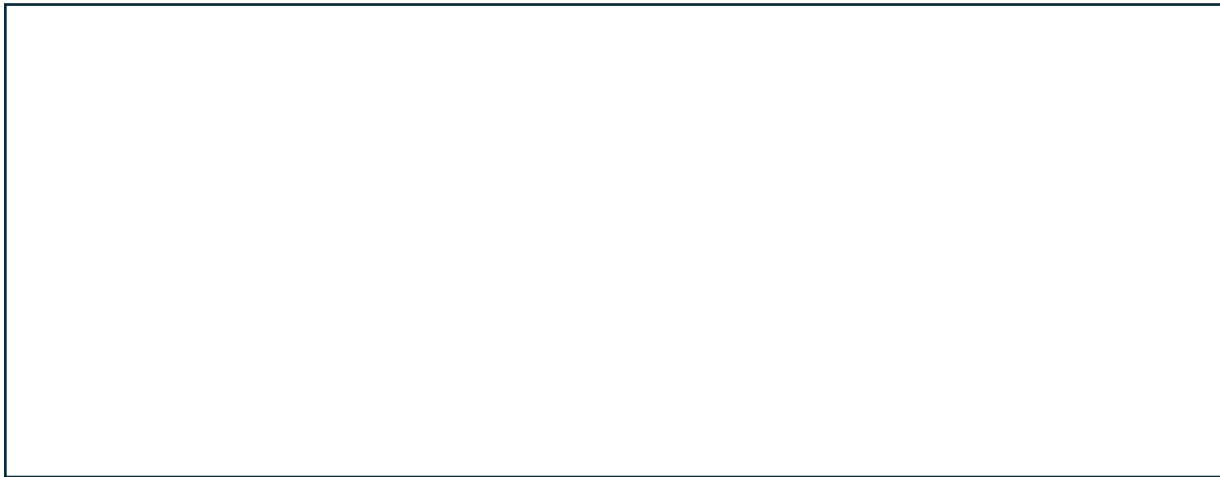
**CT6:**

```
def ct6(s):  
    t = s  
    c = chr(ord('C') - ord('A') + ord('a') + 1)  
    s += c  
    u = '-'.join(list(s))  
    return repr(u + t)  
print(ct6('XY'))
```

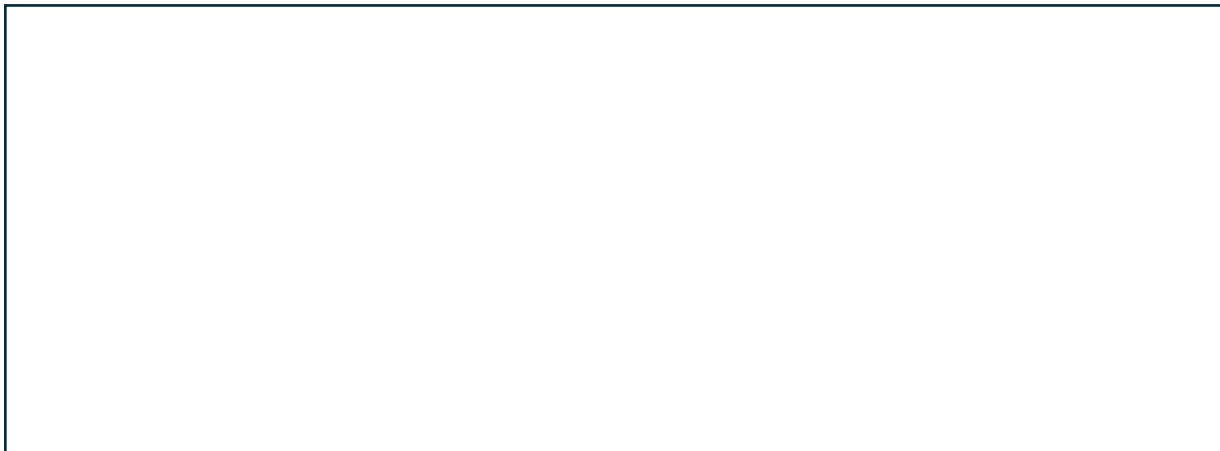


CT7:

```
def ct7(L):  
    M = copy.copy(L)  
    N = L  
    L += [5]  
    return [L, M, N]  
L = [3]  
print(ct7(L))  
print(L)
```

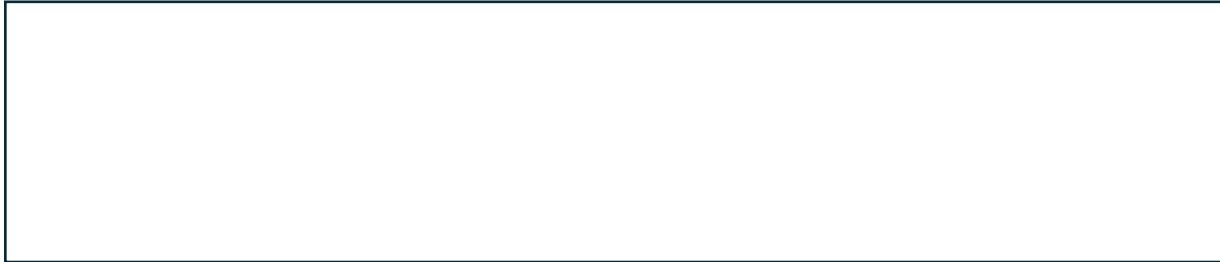
**CT8:**

```
def ct8(L):  
    return [L.index(v)**2 for v in L if v%2 == 1]  
print(ct8([2, 5, 7, 5, 2]))
```

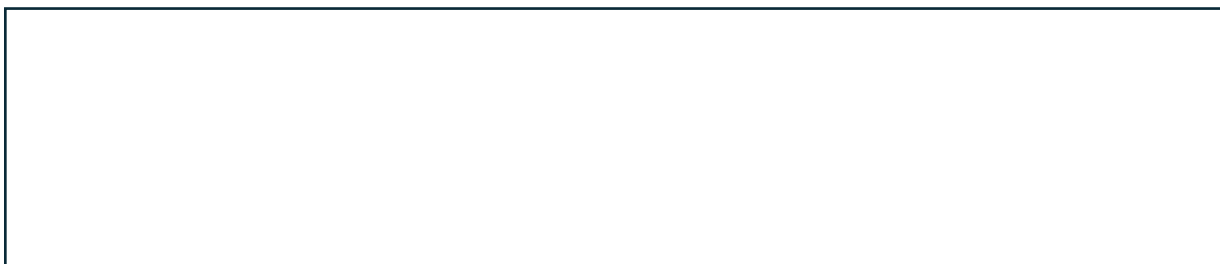


CT9:

```
def ct9(L):
    M = [ max(L), min(L) ]
    print(L.append(5))
    L = L.sort()
    M = sorted(M)
    return [L] + [0] + M
L = [2, 3]
print(ct9(L))
print(L)
```

**CT10:**

```
def ct10(L):
    M = L
    C = copy.copy(L)
    D = copy.deepcopy(L)
    M[0][0] = 3
    C.append(2)
    D[0][0] = 1
    for T in [M, C, D]: print(T)
L = [[4]]
ct10(L)
print(L)
```



Free Response / FR1: firstNDistinctSemiprimes [25 pts]

Background: a positive int x is a semiprime if x is the product of exactly two primes.

For example:

- 4 is a semiprime because $4 = 2 * 2$
- 6 is a semiprime because $6 = 2 * 3$

The first 10 semiprimes are:

- 4, 6, 9, 10, 14, 15, 21, 22, 25, 26

Next, a number x is a distinct semiprime if x is a semiprime but x is not a perfect square. Thus, 4 and 9 are not distinct semiprimes.

The first 10 distinct semiprimes are:

- 6, 10, 14, 15, 21, 22, 26, 33, 34, 35

With that in mind, write the function `firstNDistinctSemiprimes(n)` (which you may abbreviate as `fnds(n)`) that takes a possibly-negative int n and returns a sorted list of the first n distinct semiprimes. If n is negative, return `None`.

To be clear: when n is 0 return an empty list, and when n is 1 return a list of length 1.

For example:

```
assert(fnds(-1) == None)
assert(fnds(0) == [ ])
assert(fnds(1) == [ 6 ])
assert(fnds(2) == [ 6, 10 ])
assert(fnds(10) == [ 6, 10, 14, 15, 21, 22, 26, 33,
                    34, 35 ])
```

Begin your FR1 answer on the next page.

Begin your answer to FR1 here:

Continue your answer to FR1 here:

Free Response / FR2: isConcatty [25 pts]

We will say that a string s is "long" if its length is at least 4. Otherwise, the string is "short".

A possibly-non-rectangular 2d list L is "concatty" (a coined term) if:

- L is a 2d list of only strings, and among those strings:
 - No strings are empty, and
 - At least one string is a long string, and
 - Every long string is the concatenation of two short strings (where the same short string can be used twice).

For example, here is a concatty list:

```
L = [ [ 'AB', 'ABCD' ],  
      [ 'CD', 'EFG', 'EFGAB' ],  
      [ 'ABAB' ] ]
```

Here, the long strings are 'ABCD', 'EFGAB', and 'ABAB', and:

- 'ABCD' == 'AB' + 'CD', and
- 'EFGAB' == 'EFG' + 'AB'
- 'ABAB' == 'AB' + 'AB'

So L is concatty.

With that, write the function `isConcatty(L)` that takes a possibly-non-rectangular 2d list L and returns `True` if L is concatty and `False` otherwise.

Here are some test cases:

```
L = [ [ 'AB', 'ABCD' ],
      [ 'CD', 'EFG', 'EFGAB' ],
      [ 'ABAB' ] ]
assert(isConcatty(L) == True)
```

```
L = [ [ 'AB', 'CD', 'ABCD', 'CDAB' ] ]
assert(isConcatty(L) == True)
```

```
L = [ [ 'ABCD', 'CDAB' ],
      [ 'AB', 'CD' ] ]
assert(isConcatty(L) == True)
```

```
L = [ [ 'ABCD' ], [], ['AB', 'CD'] ]
assert isConcatty(L) == True
```

```
L = [ [ 'AB', 'CD', 'ABCD', '' ] ]
assert(isConcatty(L) == False) # L contains an empty string
```

```
L = [ [ 'AB', 'CD' ] ]
assert(isConcatty(L) == False) # L contains no long strings
```

```
L = [ [ 'AB', 'CD', 'CABD' ] ]
assert(isConcatty(L) == False)
# 'CABD' is not a concatenation of two short strings in L
```

```
L = [ [ 'AB', 'C', 'D', 'ABCD' ] ]
assert(isConcatty(L) == False)
# 'ABCD' is not a concatenation of *two* short strings in L
```

```
L = [ [ 'AB', 'CD', 42 ] ]
assert(isConcatty(L) == False) # 42 is not a string
```

Begin your answer to FR2 on the next page.

Begin your answer to FR2 here:

Continue your answer to FR2 here:

Free Response / FR3: Blue Dot Red Line [25 pts]

Write an app that:

- Starts with a 400x400 canvas with one blue dot of radius 20 in the center of the canvas, and a horizontal red line running from the left edge to the right edge of the canvas, 20 pixels below the bottom of the blue dot.
- The app can either be in Dot mode or in Line mode. The app starts in Dot mode. Pressing 'm' toggles between Dot and Line mode.
- When in Dot mode:
 - When the user presses the mouse, the blue dot moves to be centered on the mouse press.
- When in Line mode:
 - When the user presses the mouse, the red line remains horizontal across the entire canvas, but it moves vertically to the y position of the mouse press.
- At any time, if the line and dot intersect, then in addition to the dot and line, the app draws 'Game Over' in 40-point font centered on (200, 100).
- When the game is over, on any mouse press or key press the game will start over in its original state.

Notes:

- For anything not specified (such as the order in which you draw the dot, the line, and the 'Game Over' string), you can do anything reasonable.
- You must adhere to MVC rules. Code that violates MVC will be ignored, and will result in very large deductions.

To solve this, you must write:

- `onAppStart(app)`
- `onMousePress(app, mouseX, mouseY)`
- `onKeyPress(app, key)`
- `redrawAll(app)`

You are free to write any additional helper functions.

You do not have to write main, call `runApp`, or include imports.

Begin your answer to FR3 here:

Continue your answer to FR3 here:

Continue your answer to FR3 here:

Bonus Code Tracing (BonusCT) [Optional, 4 pts, 2 pts each]

Bonus problems are not required.

For each CT, indicate what the code prints.

Place your answer (and nothing else) in the box next to or below the code.

BonusCT1:

```
def bonusCt1(n):
    def f(n):
        x, y = n, 0
        while x: x, y = x-1, y+x
        return 2*y - n
    def g(n):
        x = 0
        while f(x) < n: x += 1
        return n - x
    return g(890)
print(bonusCt1(3))
```

BonusCT2:

```
def bonusCt2(L):
    def f(L):
        y = max([len(M) for M in L])
        M = [ [ ] for v in L ]
        for i in range(len(L)):
            for j in range(y):
                if j >= len(L[i]):
                    M[i].append(sum(L[i] + M[i]))
        return M
    return f(f(L))
```

```
L = [ [ 1, 2 ],
       [ 3, 4, 5 ],
       [ 6 ] ]
print(bonusCt2(L))
```