fullName:_____andrewID:_____
recitationLetter:_____

15-112 N22

Quiz1 version B

You **MUST** stop writing and hand in this **entire** quiz when instructed in lecture.

- You may not unstaple any pages.
- 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 this in with your paper quiz, and we will not grade it.
- Failure to hand in an intact quiz will be considered cheating. Discussing the quiz with anyone in any way, even briefly, is cheating. (You may discuss it only once the quiz has been posted to the course website.)
- You may not use any concepts (including builtin functions) we have not covered in the notes this semester. Write the word 'waffle' at the top of this page for one bonus point.
- You may not use strings, lists, indexing, tuples, dictionaries, sets, or recursion.
- We may test your code using additional test cases. Do not hardcode.
- Assume almostEqual(x, y) and roundHalfUp(n) are both supplied for you. You must write all other helper functions you wish to use.

True or False [4pts ea]

Write only the whole word "True" or "False" (and not just T or F).

_____ **TF1:** The following line of code will crash:

```
print((9 // 7 == 2) or (2 / 0 == 1))
```

_____ **TF2:** The following line of code will crash:

```
print((9 // 7 == 2) and (2 / 0 == 1))
```

_____ **TF3:** The three basic error types are runtime, prime, and syntax.

_____ **TF4:** After the following lines of code, the value of x will be an int:

```
x = 5
x += 1.0
```

CT1: Code Tracing [10pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def ct1(m):
    x = 1
    while x < 5:
        x += 2
        print('x =', x)
    for y in range(m, m+2):
        print('y =', y)
        x += y
    return x

print(ct1(2))
```

CT2: Code Tracing [10pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def f(x):
    return 2 * x - 1

def g(x):
    return f(x + 3)

def ct2(x):
    print(f(x - 2))
    x -= 2
    print(g(x))
    x = 10 % x
    return f(g(x) % 10) // 2

print(ct2(4) + 1)
```

Free Response 1: isPerfectSquare(n) [32pts]

A perfect square is any non-negative number that can be formed by the product of an integer with itself. For example, 3 * 3 == 9, so 9 is a perfect square. Also, 5 * 5 == 25, so 25 is a perfect square. Write the function isPerfectSquare(n) which takes an integer n (not necessarily positive) and returns True if it is a perfect square, and false otherwise.

Note: In addition to the restrictions on concepts we have not yet covered, you may not use loops in this problem. Solutions that use loops will lose 5 points. Here are some test cases:

```
assert(isPerfectSquare(0) == True)
assert(isPerfectSquare(1) == True)
assert(isPerfectSquare(2) == False)
assert(isPerfectSquare(9) == True)
assert(isPerfectSquare(-9) == False)
assert(isPerfectSquare(121) == True)
assert(isPerfectSquare(100000000**2) == True)
assert(isPerfectSquare(100000000**2 - 1) == False)
```

Free Response 2: nthGregorPrime(n) [32pts]

We will say that a number is a "Gregor number" (a made-up term) if it is a positive integer where the count of each digit is less than or equal to the value of the digit itself. For example, the digit 1 must not appear more than once, the digit 3 must not appear more than 3 times, and the digit 6 cannot appear more than 6 times. 0 may not appear at all. Here are some example Gregor numbers: 1, 333122, 492236134, 23213

However, these are not Gregor numbers because at least one digit appears too many times: 11, 2232, 331233, 10, 555555

With this in mind, write the function nthGregorPrime(n) that takes a non-negative int n and returns the nth number that is **both a Gregor number and prime.** You must also write the helper functions isPrime(n) and isGregorNumber(n), both of which take positive integers. You may write additional helper functions if you wish. Here are some test cases:

```
assert(nthGregorPrime(0) == 2)  # Watch out for off-by-one errors
assert(nthGregorPrime(1) == 3)
assert(nthGregorPrime(4) == 13)
assert(nthGregorPrime(5) == 17)
assert(nthGregorPrime(30) == 167)
```

You may continue your FR2 answer here, if you wish

bonusCT: Code Tracing [2pts]

This question is optional. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def f(x): return x+5
def g(x): return f(x-3)
def h(x): return g(g(x)%f(x))
def bonusCt(f, g, x):
  if (x > 0):
      return bonusCt(g, h, -f(x))
  else:
      return f(g(h(x)))
print(bonusCt(g, f, 4))
```