fullName:_____andrewID:_____ recitationLetter:_____

15-112 S24

WS1+Quiz1 version B (25 min)

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

- You may not unstaple any pages.
- 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 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 quiz, and we will not grade it.
- You may not ask questions during the quiz, except for English-language clarifications. If you are unsure how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) which we have not covered in the notes in week 1 / unit 1.
- You may not use strings, loops, lists, indexing, tuples, dictionaries, sets, or recursion.
- We may test your code using additional test cases. Do not hardcode.
- 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.

Writing Session 1 (10% of HW1)

Write setKthDigit. If you wish to use getKthDigit, you'll have to write that, too. Remember not to use loops, strings, lists, or anything else not covered in Unit 1.

Write your WS answer here

Quiz1

CT1: Code Tracing [15pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note: if a print statement would crash, just write "crash" for the output of that call. Of course, nothing else will print after that.

```
def ct1(n):
    x = n/2
    y = n//2
    print(type(y) == int)
    print(type(y - x) == int)
    print((y//n == 0) or (y//0 == n))
    print((y//0 == 0) or (y//n == 0))
    print((y//n == 0) and (y//0 == n))
    print(x, y)
ct1(5)
```

CT2: Code Tracing [15pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note: if a print statement would crash, just write "crash" for the output of that call. Of course, nothing else will print after that.

```
def ct2(x, y):
    print(x, y%10, y//10)
    x += 5
    x, y = 10*y, x//10
    print(x)
    return y

x = 86
print(ct2(x+1, 97))
print(x)
```

CT3: Code Tracing [15pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note: if a print statement would crash, just write "crash" for the output of that call. Of course, nothing else will print after that.

```
import math

def f(x):
    print(x, 'f')
    return math.ceil(x**2 / 10)

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

def ct3(x):
    print(g(x))

print(ct3(5))
```

Free Response 1: nearestPositivePowerOf10(n) [25pts]

Background: The "positive powers of 10" are the numbers 10**1, 10**2, 10**3, 10**4, and so on. The smallest positive power of 10 is 10 itself.

With this in mind, write the function `nearestPositivePowerOf10(n)` that takes a possibly-negative number n (which may be an int or float), and returns the positive power of 10 that is nearest to n, with ties going to the smaller one.

For example, to compute `nearestPositivePowerOf10(50)`, we see that 50 is 40 from 10, and 50 is 50 from 100. Since 50 is closer to 10, we return 10, so:

```
assert(nearestPositivePowerOf10(50) == 10)
```

As another example, to compute `nearestPositivePowerOf10(55)`, we see that 55 is 45 from 10, and 55 is also 45 from 100. Since it is equidistant to two positive powers of 10, we return the smaller one. So:

```
assert(nearestPositivePowerOf10(55) == 10)
```

As one last example, we see that 56 is closer to 100 than it is to 10, so:

```
assert(nearestPositivePowerOf10(56) == 100)
```

Since we only return positive powers of 10, the smallest value we can return is 10 itself. Thus:

```
assert(nearestPositivePowerOf10(-123.45) == 10)
```

Here are some additional test cases:

```
def testNearestPositivePowerOf10():
    assert(nearestPositivePowerOf10(50) == 10)
    assert(nearestPositivePowerOf10(55) == 10)
    assert(nearestPositivePowerOf10(56) == 10**2)
    assert(nearestPositivePowerOf10(550) == 10**2)
    assert(nearestPositivePowerOf10(550.1) == 10**3)
    assert(nearestPositivePowerOf10(5500) == 10**3)
    assert(nearestPositivePowerOf10(5501) == 10**4)
    assert(nearestPositivePowerOf10(-123.45) == 10)
```

Begin your FR1 answer on the following page

Write your FR1 answer here:

Free Response 2: removeZeros(v) [30pts]

Write the function `removeZeros(v)` that takes a value v, and:

- if `100 <= v <= 9999`, the function returns the same value v but with the 0 digits removed. For example, if v is 2040, the function returns 24.
- if v is any other integer, the function returns the string 'out of range'
- if v is any non-integer, the function returns 'wrong type'

Remember that you cannot use loops, strings, lists, or other concepts not covered in hw1 or week1.

```
def testRemoveZeros():
    assert(removeZeros(1204) == 124)
    assert(removeZeros(100) == 1)
    assert(removeZeros(1004) == 14)
    assert(removeZeros(1234) == 1234)
    assert(removeZeros(12345) == 'out of range')
    assert(removeZeros(99) == 'out of range')
    assert(removeZeros(100.234) == 'wrong type')
    assert(removeZeros('do not crash here') == 'wrong type')
```

Begin your FR2 answer here or on the following page

You may begin or continue your FR2 answer here, if you wish

bonusCt1 [optional, 1pt]

This problem is not required. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def f(n): return n - int(n)
def g(n): return int(100*f(n))
def bonusCt1(n): return g(g(n) - n)
print(bonusCt1(7.654))
```

bonusCt2 [optional, 1pt]

This problem is not required. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def bonusCt2(n):
    def f(n): return 10*n%(n-1)
    def g(n): return n + f(n)
    n += g(n)
    f, g = g, f
    return n + g(n)
print(bonusCt2(5))
```