fullName: _____ andrewID: _____ section: ____

15-112 F26 Quiz1 Version A
Time: 30 Minutes

You must write your name on this paper and hand this back in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. Do not unstaple any pages. All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your quiz. We will not grade anything on scrap paper.

You may not ask questions during the quiz, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the quiz, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use material we have not yet covered. Thus, do not use loops, strings, lists, tuples, sets, dictionaries, OOP, or recursion.

For all problems, assume that the required imports (such as 'import math') are present.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than the specified time noted above on this assessment.

Code Tracing [20 pts, 10 pts each]

CT1: [10 pts]

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
def g(x):
    return 10 * x if x <= 100 else x / 10

def f(x):
    if x // 10 % 10 <= 5:
        x += g(x % 10)
    return g(x)

print(f(123))
```

CT2: [10 pts]

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
def g(x): return math.floor(x ** 0.5)

def h(x): return math.ceil(x) - math.ceil(-x)

def m(f, x):
    d = int(x) % 10
    x /= 10
    return f(x) + d

x = 95
print(m(g, x), m(h, x))
```

Fill in the Blank [20 pts, 10 pts each]

FitB1: [10 pts]

Fill in the blank so f(n) == g(n) for all integers n.

```
def f(n):
    return (n % 5 != 0)

def g(n):
    return n // 5 * 5 != _____________________________________________
```

FitB2: [10 pts]

Fill in the blank so f(n) == g(n) for all integers n.

```
def f(n):
    if (n < 0) or (n >= 10**4):
        return False
    ab = n // 100
    cd = n % 100
    return ab >= cd

def g(n):
    if (n < 0) or (n >= 10**4):
        return False
    m = n
    m //= 100
    n %= 100
    return max(m, n) == _____________________________________________
```

FR0: uniqueDigits [half credit for FR1 or FR2]

Important note: Only attempt this problem (for half-credit) if you cross out your answer (with a large 'X' across the whole answer box) to one of the other two FRs (and then we will not grade the crossed-out FR answer). If you do not cross out one of the other two FR answers, we will not grade this problem even if you complete it.

Write the function uniqueDigits(n) that takes a possibly-negative integer n. If n is a positive number with 3 digits, return the number of unique digits in n. If n is non-positive or does not have exactly 3 digits, return 'out-of-range'.

For example, if n is 585, there are 2 unique digit (5 and 8), so:

```
assert(uniqueDigits(585) == 2)
```

Here are a few more examples:

```
assert(uniqueDigits(587) == 3)
assert(uniqueDigits(555) == 1)
assert(uniqueDigits(99) == 'out-of-range')
assert(uniqueDigits(1234) == 'out-of-range')
assert(uniqueDigits(-587) == 'out-of-range')
```

Write your solution on the next page.

FR1: perfectSquaresInRange [30 pts]

Background: the perfect squares are the integers 0, 1, 4, 9, 16,...

With that, write the function perfectSquaresInRange(lo, hi), which we will abbreviate as psir(lo, hi), that takes two possibly-negative integers lo and hi, and returns the number of perfect squares in the range [lo, hi] (so lo and hi are included in the range). If hi < lo, return 0.

For example, if lo is 0 and hi is 4, the perfect squares in the range [0, 4] are 0, 1, and 4, so:

```
assert(psir(0, 4) == 3)   # 0, 1, 4
```

Here are a few more examples:

```
assert(psir(1, 4) == 2)   # 1, 4
assert(psir(5, 8) == 0)   # none in range
assert(psir(15, 35) == 2) # 16, 25
assert(psir(15, 36) == 3) # 16, 25, 36
assert(psir(-5, 0) == 1)  # 0
assert(psir(-5, -1) == 0) # none in range
assert(psir(16, 16) == 1) # 16
assert(psir(4, 0) == 0)   # hi < lo
```

Write your solution on the next page.

FR2: dominantParity [30 pts]

We will say that the 'dominant parity' (a coined term) of an integer x is:

- 'even' if the majority of the digits in x are even,
- 'odd' if the majority of digits in x are odd, and
- 'neutral' otherwise.

With that, write the function dominantParity(x) that takes a value x and returns:

- the dominant parity of x if x is an integer between -99 and +99 (inclusive),
- 'out-of-range' if x is an integer outside that range, or
- 'not-an-int' if x is not an integer.

Here are some test cases for you:

```
assert(dominantParity(0) == 'even')
assert(dominantParity(1) == 'odd')
assert(dominantParity(-1) == 'odd')
assert(dominantParity(24) == 'even')
assert(dominantParity(98) == 'neutral')
assert(dominantParity(99) == 'odd')
assert(dominantParity(-99) == 'odd')
assert(dominantParity(100) == 'out-of-range')
assert(dominantParity(24.) == 'not-an-int')
```

Recall that you cannot use strings or lists here.

Write your solution on the next page.

BonusCT

These CTs are optional, and intended to be very challenging. They are worth very few points. Indicate what the following code prints. Place your answers (and nothing else) in the boxes below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1: [1pt]

```
def bonusCt1(x):
    def f(x, y): return x ** y == y ** x
    def g(x, n): return x // 10 ** n % 10
    def h(x, n):
        if f(g(x, n), g(x, n//2)):
            print(n, end='')
    # The next line does not crash.  It calls h 5 times.
    h(x,4);h(x,3);h(x,2);h(x,1);h(x,0)
bonusCt1(63245)
```

bonusCT2: [1pt]

```
import math
def bonusCt2(n):
    def f(n):
        return 0 if n < 10**-5 else n + f(n/2)
    return math.floor(f(n))
print(bonusCt2(1234))
```