fullName: _____ andrewID: _____ section: ____

15-112 F26 Midterm1 Version A
Time: 80 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 exam, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the exam, 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 exam. All syllabus policies apply.

You may not discuss this exam 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 sets, dictionaries, OOP, or recursion. Also: do not use eval() or exec().

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

You may not leave the exam room during the final 15 minutes of this exam. If you finish early, there are ample bonus problems to keep you gainfully occupied.

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.

True or False [10 pts, 1 pt each]

Indicate True or False for each of the following. Indicate your answer by filling in the dot.

1. If `(x is y)` is True, then `(x == y)` is True.

○ True

○ False

2. For all positive integers x and y, `(x % y < y)` is True.

○ True

○ False

3. For all non-zero integers x and y, `((x // y) == int(x / y))` is True.

○ True

○ False

4. If `x = (1)` then `(type(x) == tuple)` is True.

○ True

○ False

5. For all positive integers x and y, `(len(range(x, y, y)) < 2)` is True.

○ True

○ False

6. For any string c, `((len(c) != 1) or (c == chr(ord(c))))` is True.

○ True

○ False

7. For any nonempty string s, `s[-len(s):2*len(s)]` crashes.

○ True

○ False

8. For any string s, `repr(s)` can help to tell if s is empty or contains some spaces.

○ True

○ False

9. For any string s, `s.strip()` removes all the whitespace from s.

○ True

○ False

10. Function parameters are aliases and not copies in Python.

○ True

○ False

Code Tracing [30 pts, 3 pts each]

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.

```
# CT1:

import copy

def f(x, y):
    x += y
    return x + y

def ct1(x, y):
    return f(x, y) if x < 5 else f(f(x, 2), y)

print([ct1(x, 3) for x in range(3, 7, 2)])
```

```
# CT2:

def ct2(n):
    L = [ ]
    for x in range(n):
        if x % 2 == 0:
            continue
        for y in range(x, n):
            if x + y == 4:
                break
            L.append((x, y))
    return L

print(ct2(5))
```

```
# CT3:

def ct3(s):
    L = [ ]
    for c in s:
        if c.isdigit():
            L.append(chr(ord('a') + int(c)))
    return '-'.join(L)

print(ct3('I have 2 cats and 13 birds.'))
```



```
# CT4:

def g1(s):
    return s.islower()

def g2(s):
    return (s != '') and (s[-1].isdigit())

def ct4(g, s):
    L = [ ]
    for i in range(1, len(s)+1):
        L.append(int(g(s[:i])))
    return L

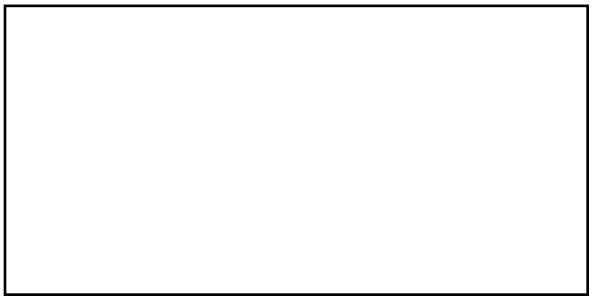
s = 'a1B2'
print(ct4(g1, s))
print(ct4(g2, s))
```



```
# CT5:

def ct5(L, d):
    M = L
    N = 2 * [2*v for v in L]
    M.insert(0, N.pop(0))
    L = L + [d]
    M[1] += sum(L[:1])
    M = sorted(M)[1:]
    print(f'L={L}')
    print(f'M={M}')
    print(f'N={N}')

L = [3, 4]
ct5(L, 5)
print(f'L={L}') # don't miss this
```



```
# CT6:

def ct6(L):
    try:
        for i in range(len(L)):
            if L[i] % 2 == 0:
                L.pop(i)
    except:
        print(f'i={i}')

L = [2,3,4,5]
ct6(L)
print(f'L={L}') # don't miss this
```



```
# CT7:

def ct7(L):
    for i in range(len(L)):
        M = L[i:-i]
        if M != []:
            print(M)

ct7([9,8,7,6,5])
```



```
# CT8:

def ct8(L):
    M = [ ]
    for s in L:
        t = s[len(M):]
        M += list(t)
    return '-'.join(M)

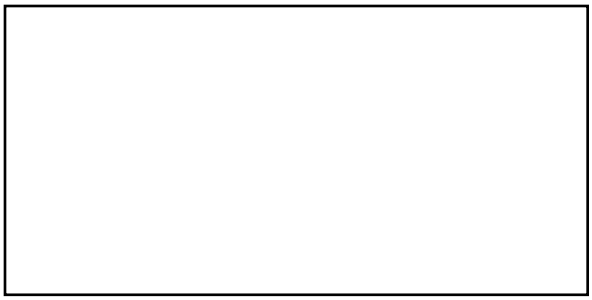
print(ct8(['AB', 'C', 'DEF', 'G', 'HIJK', 'L']))
```



```
# CT9:

def ct9(L):
    M = copy.copy(L)
    N = copy.deepcopy(L)
    M.append(N[0].pop())
    N.extend(M.pop(0))
    L = L + [[4]]
    M += [sum(L[0])]
    print(f'L={L}')
    print(f'M={M}')
    print(f'N={N}')

L = [[1, 2]]
ct9(L)
print(f'L={L}')  # don't miss this
```



```
# CT10:

def ct10(L):
    rows, cols = len(L), len(L[0])
    for row in range(rows):
        for col in range(cols):
            L[0][col] += L[row][col]

L = [[2, 3], [5, 7]]
ct10(L)
print(L)  # don't miss this
```



Standard Fill in the Blank [15 pts, 3 pts each]

For each question, fill in the blank so the functions work as described. See the test functions for examples. Do not hardcode to the test functions.

Standard-FitB1:

```
def reverseNonNegativeInt(n):
    # This function takes a non-negative int n and returns the
    # number formed by reversing the digits in n.
    assert(n >= 0)
    result = 0
    while n > 0:
        result = ____________________________________________________
        n //= 10
    return result

def testReverseNonNegativeInt():
    assert(reverseNonNegativeInt(123) == 321)
    assert(reverseNonNegativeInt(450) == 54)
```

Standard-FitB2:

```
def removeEmptyLines(s):
    # This function takes a multi-line string s and returns the
    # multi-line string formed by removing all the empty lines in s.
    L = [ ]
    for line in s.splitlines():
        if line != '':
            L.append(line)

    return ____________________________________________________

def testRemoveEmptyLines():
    assert(removeEmptyLines('\nAB\n\nC\n\n') == 'AB\nC')
    assert(removeEmptyLines('\n\n\n\n\n') == '')
```

Standard-FitB3:

```
def removeLargestInts(L):
    # This function takes a list L of arbitrary values that contains
    # at least one int, and non-mutatingly returns a new list formed
    # by removing every occurrence of the largest int in L.

    largestInt = None
    for v in L:

        if _____________________________________________________________:

            if (largestInt == None) or (v > largestInt):
                largestInt = v
    M = [ ]
    for v in L:
        if v != largestInt:
            M.append(v)
    return M

def testRemoveLargestInts():
    L = [1, 'cat', 2, 'dog', 2, 1]
    assert(removeLargestInts(L) == [1, 'cat', 'dog', 1])
    # confirm non-mutating
    assert(L == [1, 'cat', 2, 'dog', 2, 1])
```

Standard-FitB4:

```
def evalExpr(s):
    # This function takes a non-empty string that is a legal
    # expression containing integers separated by ' + ' or ' - '
    # and evaluates that expression.
    result = 0
    sign = 1
    for part in s.split():
        if part == '+':
            sign = +1
        elif part == '-':
            sign = -1
        else:
            result += _____________________________________________
    return result

def testEvalExpr():
    assert(evalExpr('2 + 10 - 3') == 9)
    assert(evalExpr('2 + 10 - 12 - 42') == -42)
```

Standard-FitB5:

```
def addIntegerAveragesRow(data):
    # This function takes a 2d list data where the first row contains
    # labels ('Student' in the first column, followed by quiz names in
    # the rest of the columns), and each row after that contains a
    # student's name and scores.
    # The function mutatingly inserts a row of integer averages right
    # after the first row. The new row's first value is 'Average', and
    # each value after that is average of the students' scores for
    # that quiz.

    rows, cols = len(data), len(data[0])
    newRow = [ 'Average' ]
    for col in range(1, cols):
        values = [data[row][col] for row in range(1, rows)]
        average = round(sum(values) / len(values))
        newRow.append(average)

    # Fill in the next line (the last line of the function):

    ____________________________________________________________________


def testAddIntegerAveragesRow():
    data = [['Student', 'Quiz1', 'Quiz2'],
            ['Fred',         80,      60],
            ['Wilma',        90,     100]]
    assert(addIntegerAveragesRow(data) == None)
    assert(data == [
            ['Student', 'Quiz1', 'Quiz2'],
            ['Average',      85,      80], # <-- this row was added
            ['Fred',         80,      60],
            ['Wilma',        90,     100]])
```

New-Style Fill in the Blank [5 pts, 2.5 pts each]

New-Style FitB1:
Fill in the blank so f(n) == g(n) for all integers n.

```
def f(n):
    n = abs(n)
    r = 0
    while n > 0:
        d = n % 10
        n //= 10
        if d > r:
            r = d
    return r

def g(n):
    n = abs(n)
    L = list(str(n))

    return _____________________________________________________________
```

New-Style FitB2:
Fill in the blank so f(n) == g(n) for all integers n.

```
def f(n):
    if n <= 0:
        return 'out-of-range'
    L = [n]
    for counter in range(1, n):
        L.append(L[-1] * n)
    return L.pop()

def g(n):
    if n <= 0:
        return 'out-of-range'

    return _____________________________________________________________
```

FR1: mutatingMoveIntsToFront [15 pts]

Write the function mutatingMoveIntsToFront(L) that takes a list L of arbitrary values and mutates L so that all the integers in L are moved to the front (while still in the same order), and all the non-integers in L are moved to the back (while still in the same order). As with most mutating functions, this function should return None.

For example, if `L = [1, 'dog', 2, 0.1, 'cats']`:

- The integers in L are `[1, 2]` and these are moved to the front of L, in that order.
- The non-integers in L are `['dog', 0.1, 'cats']` and these are moved to the back of L, in that order.

Thus:

```
L = [1, 'dog', 2, 0.1, 'cats']
assert(mutatingMoveIntsToFront(L) == None)
assert(L == [1, 2, 'dog', 0.1, 'cats'])
```

Here are a few other examples:

```
L = [42, 5, 12]
assert(mutatingMoveIntsToFront(L) == None)
assert(L == [42, 5, 12])

L = ['dog', 'cats']
assert(mutatingMoveIntsToFront(L) == None)
assert(L == ['dog', 'cats'])

L = [ ]
assert(mutatingMoveIntsToFront(L) == None)
assert(L == [ ])
```

Write your solution on the next page.

FR2: getBreedCounts [25 pts]

Reminder: you may not use dicts on this exam.

Background: This problem uses a log of dog breed sightings, which is a non-empty multiline string like so:

```
    log = '''\
4 labs
14 pugs
1 hound
1 pug'''
```

Notes:

- The log contains no empty lines or extra spaces.
- The log contains at least one line.
- Every line has a positive integer followed by the breed.
- All letters are lowercase.
- Any breed ending in 's' is plural (like 'pugs') and any breed not ending in 's' is singular (like 'pug').
- The log contains no errors.

With that, write the function getBreedCounts(log) that takes a log as described and returns a single string of the total counts of each breed in alphabetical order, separated by a comma and a space, like so:

```
    assert(getBreedCounts(log) == '1 hound, 4 labs, 15 pugs')
```

Note that 'hound' is singular because there is only one hound, but the other breeds are plural because there is more than one of those breeds.

Hint: we found it very helpful to start with these two lists:

```
    breeds = [ ]
    counts = [ ]
```

We maintained the lists so breeds[i] is the name of a breed (like 'pug') and counts[i] is the count for that breed. At the end, we converted these two lists into the output format.

Write your solution on the next page.

You may continue your solution on the next page if needed.

BonusCT [5 pts, 1 pt each]
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:

```
import string

def bonusCt1(s, d):
    for c in s*d:
        d += ord(c) - ord(s[0])
    return d

print(bonusCt1(string.ascii_uppercase, 3))
```

bonusCT2:

```
def bonusCt2(n):
    L = list(range(n+1))
    M = [ ]
    for i in range(2, len(L)):
        if L[i] > 0:
            M.append(i)
            for k in range(i, len(L), i):
                L[k] = -abs(L[k])
    return [v for v in M if 2*v > n]

print(bonusCt2(20))
```

bonusCT3:

```
def bonusCt3(k):
    k0 = k
    def f(z): return f(z-1) + 2*z - 1 if z else 0
    while k < 75: k = f(k)
    return k - k0

print(bonusCt3(3))
```



bonusCT4:

```
def bonusCt4(k):
    L = [[0]*3 for _ in range(3)]
    for i in range(3): L[i][(i+k)%3] = 10*i
    z = 7
    while z:
        M = L[0]
        if 0 in M:
            M[M.index(0)] = z
            z -= 1
        L.append(L.pop(0))
    return L[0]

print(bonusCt4(5))
```



bonusCT5:

```
def bonusCt5(d):
    def f(d):
        L = [0]
        while L[-1] < 1: L.append(L[-1]+d)
        return L
    t = u = 0
    for x in f(d):
        for y in f(d):
            u += 1
            if x**2 + y**2 <= 1:
                t += 400
    return round(t/u)

print(bonusCt5(0.001))
```



This page is intentionally blank.