fullName:_____ andrewID:_____ section:___

15-112 S25
Quiz1 version A

Read these instructions carefully before starting:

1. Quiz versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. 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.
3. Do not discuss the quiz with anyone else until after 5pm.
    - This applies to everyone, including students in either lecture.
4. Do not use your own scrap paper.
    - You should not need scrap paper, there is plenty of room for you on the quiz.
    - However, if you absolutely must use scrap paper, raise your hand and we will provide some. Then, you must write your andrew id clearly on the scrap paper, and hand in the scrap paper with your paper quiz. We will not grade anything on your scrap paper.
5. You may not ask questions during the quiz.
    - The one exception is for English-language clarifications.
    - If you are unsure how to interpret a problem, just take your best guess.
6. Do not use any concepts (including built-in functions) not covered in the notes through week 1 / unit 1.
    - Do not use strings, loops, lists, tuples, dictionaries, sets, or recursion.
7. Do not hardcode your solutions.
    - We may test your code using additional test cases.
    - Hardcoding will receive zero points.
8. 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.
9. Good luck!

Code Tracing (CT) [20 pts, 5 pts each]
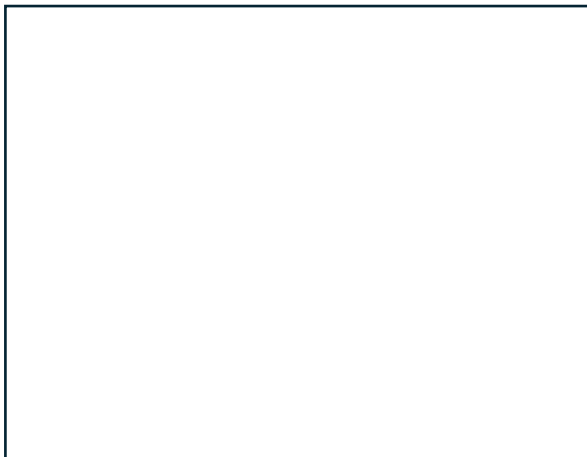For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

Note: all the floats that are printed in these CTs have no more than one digit after the decimal point.

CT1:
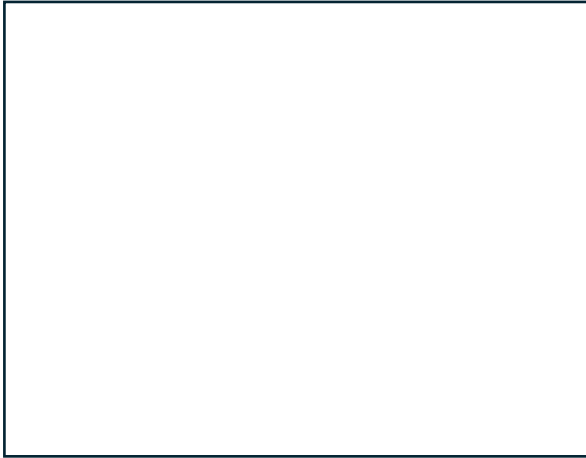
```python
def ct1(a, b):
    print(a % b, a % 2 * b)

print(ct1(7, 3))
ct1(3, 7)
```

CT2:

```python
def ct2(m, n):
    x = m ** n
    x //= n
    x /= m
    return x

print(ct2(2, 5))
```
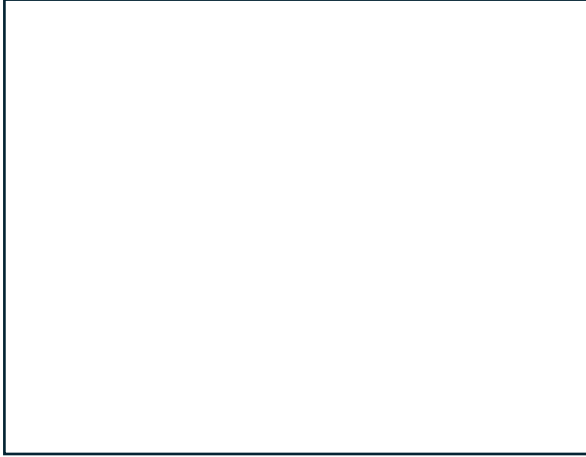
CT3:

```
def ct3(x, y):
    print(x or y, x and y)

ct3(0, 1)
ct3(2, 3)
```
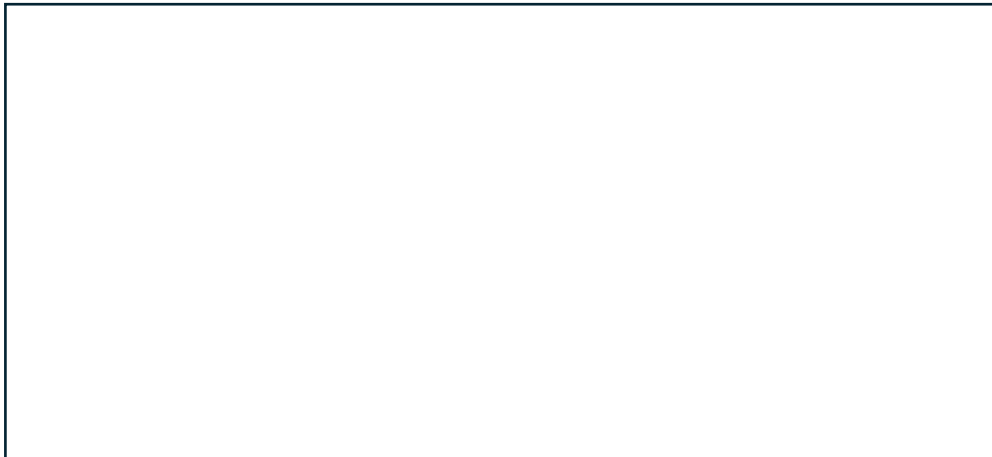
CT4:

```python
def ct4(x, d):
    if x < 0:
        x **= 2
    else:
        x = int(x)
    d += 1 if x % 2 == 0 else 2
    return f'{x} + {d} = {x+d}'

print(ct4(-3, 3))
print(ct4(4.9, 2.1))
```

Free Response / FR1: distanceToNearestTenth [30 pts]

Write the function `distanceToNearestTenth(n)` that takes a possibly-negative float n, and returns the positive distance from n to the nearest tenth.

For example, if n `== 4.08`, the nearest tenth to n is 4.10, and the distance from 4.08 to 4.10 is 0.02, so
`distanceToNearestTenth(4.08) == 0.02`. Actually, since these are floats, we will use `almostEqual` to check for correctness.

Thus, here are some test cases:
```
    assert(almostEqual(distanceToNearestTenth(1.1), 0))
    assert(almostEqual(distanceToNearestTenth(4.08), 0.02))
    assert(almostEqual(distanceToNearestTenth(2.71), 0.01))
    assert(almostEqual(distanceToNearestTenth(3.35), 0.05))
    assert(almostEqual(distanceToNearestTenth(-4.08), 0.02))
```

Remember not to use loops, lists, strings, or anything we have not covered in the week 1 notes.

Begin your FR1 answer on the next page.

Begin your answer to FR1 here:

Free Response / FR2: isAddish [50 pts]

Background: we will say that a positive integer n is "addish" (a coined term) if its `digitCount` is 3*k for some positive integer k, and that n can be split into 3 smaller integers a, b, and c (each with a `digitCount` of k), such that `a + b == c`. These integers should be assigned so that a contains the leftmost digits, b contains the middle digits, and c contains the rightmost digits. For example:

- If n is 279, n has 3 digits, which is 3*k when k=1. So, we can split n into 3 k-digit numbers a=2, b=7, and c=9. Because `2 + 7 == 9`, we know that `a + b == c`, so 279 is addish.

- If n is 972, then a=9, b=7, and c=2. This is not addish because `9 + 7 != 2`, so `a + b != c`.

- If n is 127183 (6 digits), we can split n into a=12, b=71, and c=83, and `12 + 71 == 83`, so `a + b == c`, so 127183 is addish.

- If n is 12324, it has only 5 digits. 5 is not a multiple of 3, so n is not addish.

With that in mind, write the function `isAddish(n)` that takes an arbitrary Python value (perhaps not an int, and if an int, perhaps not positive), and returns `True` if n is addish and `False` otherwise.

Remember not to use loops, lists, strings, or anything we have not covered in the week 1 notes. You may assume that `digitCount(n)` is already written for you.

Here are some test cases for you:
```
assert(isAddish(279) == True)              # 2 + 7 == 9
assert(isAddish(972) == False)             # 9 + 7 != 2
assert(isAddish(127183) == True)           # 12 + 71 == 83
assert(isAddish(123400011235) == True)     # 1234+1 == 1235
assert(isAddish(278) == False)             # 2 + 7 != 8
assert(isAddish(-279) == False)            # not positive
assert(isAddish(0) == False)               # not positive
assert(isAddish(279.0) == False)           # not an int
assert(isAddish('279') == False)           # not an int
```

Begin your answer to FR2 here:

Bonus Code Tracing (BonusCT) [Optional, 2 pts]
Bonus problems are not required.
For this CT, indicate what the code prints.
Place your answer (and nothing else) in the box below the code.

BonusCT1:

```python
def f(g, h, x): return g(h(x)) + h(g(x))
def g(h): return h**2 + h
def h(g): return g**3 - g + 1
def bonusCt1():
    print(f(g,h,1) + f(h,g,2))

bonusCt1()
```