fullName:_____ andrewID:_____ section:___

15-112 F25
Practice Midterm1

This is the Practice Midterm1, part of Prep 3. In order to receive credit, you must simulate taking this as if it was an actual quiz, and you **must write your name on this paper and hand this back** in during recitation on 9/10. This Practice Midterm is graded based on completion and effort rather than correctness, but if we do not receive it immediately, or if you have not demonstrated apparent effort, you will receive a zero on the assessment.

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

Do not use lists, tuples, sets, dictionaries, recursion, or anything else disallowed in units 1 and 2.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than 80 minutes on this assessment.

Code Tracing

CT1[6 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.

```
import math

def ct1(x, y, z):
    print(x ** x - y / x + y * x - z)
    y += max(x,y,z) // min(x,y,z)
    z %= x
    return 100*x + 10*y + z
print(ct1(2, 3, 5))
```

CT2[6 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 f(x): return 1+2*x
def g(x): return x//2
def h(x): return f(g(x)) if (x%2 == 0) else g(f(x))
def ct2(x):
    print(h(x))
    print(h(x+1))
print(ct2(5))
```

CT3[6 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 ct3(x, y):
    for z in range(y, x):
        print('A', z)
        break
    for z in range(y, x, -1):
        print('B', z)
        break
    for z in range(x, y):
        if (z < y//2):
            if (z%2 == 0): print ('C', z, end=' ')
            elif (z%5 == 0): print('D', z, end=' ')
        elif (x+y+z == 27):
            print('E', z)
    w = 0
    for z in range(x, 10*x, x):
        if (z < 5*x): continue
        elif (z >= 7*x): return w
        w += z
    return 99
print(ct3(3, 13))
```

CT4[6 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 ct4(x):
    counter = 0
    target = 2 # does not start at 0
    while (target < 4):
        if ((x + counter//2) >= target):
            print(counter, target, end=' ')
            target += 1
        counter += 1
    return 10*counter+target
print(ct4(2)) # prints 5 values
```

CT5[6 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 ct5(s):
    #   Hint: ord('A') > ord('9')
    t = 'Z'
    for c in s.upper():
        if (c > t[-1]):
            u = c
        elif (c >= 'A'):
            u = str(ord(c) - ord('A'))
        else:
            u = 'n' + c
        t += u
    return t
print(ct5("a1B2"))
```

CT6[6 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.

```
import math
def ct6(x):
    y, z = math.ceil(x), math.floor(x)
    print(y, z)
    if (y < 0) and (z/0 < 0):
        print('yes')
print(ct6(2.3))
print(ct6(-2.3))
```

Free Response

Your functions should work generally for the kinds of inputs specified in the problem statement, and we may test your code using additional test cases. In graded exams, we will manually grade both of these problems for partial credit if you do not pass all the test cases. **Do not use lists, tuples, sets, dictionaries, recursion, or anything else disallowed in units 1 and 2.**

FR1[16pts]: isAlmostSquare(n)

Write the function `isAlmostSquare(n)` that takes any Python value and returns `True` if it is an int within 1 (inclusive) of a perfect square, and `False` otherwise. Be sure to return `False` for non-int values of n.

Here are some test cases for you:

```
assert(isAlmostSquare(25.0) == False) # float
assert(isAlmostSquare('25') == False) # string

assert(isAlmostSquare(23) == False)
assert(isAlmostSquare(24) == True)
assert(isAlmostSquare(25) == True) # 5**2
assert(isAlmostSquare(26) == True)
assert(isAlmostSquare(27) == False)

assert(isAlmostSquare(-2) == False)
assert(isAlmostSquare(-1) == True)
assert(isAlmostSquare(0) == True) # 0**2

assert(isAlmostSquare(98) == False)
assert(isAlmostSquare(99) == True)
assert(isAlmostSquare(100) == True) # 10**2
assert(isAlmostSquare(101) == True)
assert(isAlmostSquare(102) == False)

assert(isAlmostSquare(100000003**2 - 1) == True)
assert(isAlmostSquare(100000003**2 - 2) == False)
```

Write your answer on the following page

FR2[16pts]: nthLargerTwinPrime(n)

Background: First, here are the first several primes:
`2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,...`

Now, a "larger-twin-prime" is a prime that is 2 larger than the next-smaller prime. For example, the first larger-twin-prime is 5, because 5 is 2 larger than 3. Here are the first several larger-twin-primes:
`5, 7, 13, 19, 31, 43,...`

With that in mind, write the function `nthLargerTwinPrime(n)` that takes a non-negative int n and returns the nth larger-twin-prime.

Here are some test cases for you:

```
assert(nthLargerTwinPrime(0) == 5)
assert(nthLargerTwinPrime(1) == 7)
assert(nthLargerTwinPrime(2) == 13)
assert(nthLargerTwinPrime(3) == 19)
assert(nthLargerTwinPrime(4) == 31)
assert(nthLargerTwinPrime(5) == 43)
```

Begin your answer here or on the following page

FR3[16pts]: sameOddestDigit(m, n)

Background: Say that the 'oddest' digit of a number (a coined term) is the odd digit that occurs the most in that number, with ties going to the larger digit, or 0 if the number has no odd digits.

For example, the oddest digit of 123454321 is 3. Similarly, the oddest digit of -123123 is also 3.

With this in mind, write the function `sameOddestDigit(m, n)` that does not use strings and that takes two possibly-negative int values and returns `True` if they have the same oddest digit and `False` otherwise.

Here are some test cases for you:

```
    assert(sameOddestDigit(123454321, -123123) == True)  # 3 and 3
    assert(sameOddestDigit(123454321, -123122) == False) # 3 and 1
    assert(sameOddestDigit(2468, 24689) == False) # 0 and 9
    assert(sameOddestDigit(2468, 20000) == True)  # 0 and 0
```

Reminder: you may not use strings here.

Begin your answer here or on the following page

FR4[16pts]: makeEdits(spec)

Background: This problem uses a string called `'spec'` which is a string of lowercase letters followed by editing commands on that string, separated by dots (.). There are two kinds of editing commands:

- `'1cd'` means replace all c's with d's, and
- `'2e'` means delete all the e's.

Note that replace commands are always a `'1'` followed by two characters, and delete commands are always a `'2'` followed by one character.

For example:
`'abcabde.1ax.2b'`

Here, we make the edits on this spec as follows

1. Start with `'abcabde'`
2. Replace all a's with x's to get `'xbcxbde'`
3. Delete all b's to get `'xcxde'`

With this in mind, write the function `makeEdits(spec)` that takes a spec as just described and returns the string that results from making the edits in the spec.

Here are some test cases for you:

```
assert(makeEdits('abcabde.1ax') == 'xbcxbde')
assert(makeEdits('abcabde.1ax.2b') == 'xcxde')
assert(makeEdits('abcabde.1ax.2b.1xy') == 'ycyde')
```

Note: while you may use the `split` method, you may not store the result of that in a list, but you can loop over its result.

Write your answer on the following page

You may continue writing your FR4 answer here if you wish