fullName:_____ andrewID:_____ section:___

15-112 S25
Quiz2

This is the paper version of Quiz2. You **must write your name on this paper and hand this back** in immediately after the assessment, even if you do not use it to write your answers. If we do not receive it immediately, you will receive a zero on the assessment.

Quiz2 is located in section 2.10 of CS Academy (the end of Unit 2), and contains two parts. Part 1 will become visible once you are allowed to begin. Once the time for Part 1 has elapsed, we will close Part 1 and open Part 2.

We will not grade anything you write on these pages unless you initial the box below:

_____ **Write your initials here if and only if you wish for us to grade what you have written on this paper *instead of* any answers submitted in CS Academy.** Leave the space blank if you wish for us to grade what is written in CS Academy (the default, recommended option). Note: Even if you write your answers on paper, you may not run any code related to the Part 1 problems (MCs and CTs).

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 TAs or faculty during the assessment. All syllabus policies apply.

Some students will take this at a different time with testing accommodations. As such, 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 lists, sets, dictionaries, recursion, or anything else disallowed in the original problem.

Do not open this or look inside (even briefly) before we instruct you to begin. Close it once you are done.

Part 1 [40pts total]: Multiple Choice and Code Tracing

MC1[4 pts]: What will happen if you run the following code?

```
x = 20
while True:
    x += 3
    if x == 32:
        break
print(x)
```

- ◯ a. It will print `30`
- ◯ b. It will print `32`
- ◯ c. It will loop forever
- ◯ d. It will crash due to a syntax error

MC2[4 pts]: Which loop prints the values 5, 4, 3, 2, 1, 0 in that order, each on their own line?

- ◯ a.
```
for i in range(5, 0, -1):
    print(i)
```
- ◯ b.
```
for i in range(5, -1, -1):
    print(i)
```
- ◯ c.
```
for i in range(6, 0, -1):
    print(i)
```
- ◯ d.
```
for i in range(6, -1, -1):
    print(i)
```

MC3[4 pts]: What will happen if you run the following code? Look carefully!

```
def foo(s):
  s.replace(' ', '-')
  return s
print(foo('a b c d'))
```

- ◯ a. It will print the string `'a b c d'`
- ◯ b. It will print the string `'a-b c d'`
- ◯ c. It will print the string `'a b c-d'`
- ◯ d. It will print the string `'a-b-c-d'`
- ◯ e. It will crash due to a runtime error

CT1[10 pts]:
Indicate what the following code prints. Place your answer (and nothing else) in the box below.

```
# On this quiz and in general, CTs run without crashing
def ct(s):
    r = ""
    for i in range(len(s)):
        c = s[i]
        if c.isupper():
            c = chr(ord(c) + i)
        elif c.islower() and c not in 'turtle':
            c = chr(ord(c) + 2)
            c = c.upper()
        r = c + r
    return r

print(ct("catDAD?"))
```

CT2[10 pts]:
Indicate what the following code prints. Place your answer (and nothing else) in the box below.

```
# On this quiz and in general, CTs run without crashing
def ct(s):
    r = ""
    while (len(s) > 2):
        r += s[2 : 4] + "-"
        s = s[1 : len(s)-1 : 2]
        print(s)
    return r + s
print(ct("abcd123"))
```

Part 2[60pts total]: 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. We will manually grade both of these problems for partial credit if you do not pass all the test cases.

FR1[30pts]: nthThreeish(n)

We will say that an integer n is threeish (a made-up term) if:

- `n >= 100`
- and the difference between each consecutive (i.e. neighboring) digit in n is 3

Consider the number n = 1474:

- `1474 >= 100`
- and the difference between each consecutive digit is:
  - `abs(1 - 4) == 3`
  - `abs(4 - 7) == 3`
  - `abs(7 - 4) == 3`

So we see that 1474 is threeish.

Here are the first 10 threeish numbers: 141, 147, 252, 258, 303, 363, 369, 414, 474, 525

With that, write the function **`nthThreeish(n)`** that takes a non-negative integer n, and returns the nth threeish number, so `nthThreeish(0)` returns 141.

Note: You may use strings in this problem if you wish, though we do not necessarily recommend it. As usual (and regardless of whether the autograder stops you), you may *not* use concepts we have not yet covered in lecture, like lists.

Hint: We strongly recommend that you write a helper function for this problem. It will make debugging much easier.

Here are some test cases:

```
assert(nthThreeish(0) == 141)
assert(nthThreeish(1) == 147)
assert(nthThreeish(2) == 252)
assert(nthThreeish(3) == 258)
assert(nthThreeish(9) == 525)
assert(nthThreeish(30) == 4147)
assert(nthThreeish(50) == 14147)
```

Define nthThreeish(n) on the next page

```
# Define nthThreeish(n) below:
```

```
# You may continue your answer here if you wish
```

FR2[30 pts]: simpleEval(s)

Write the function simpleEval(s) which takes a non-empty string consisting of only digits, single spaces, '+', '-', and '.' and evaluates the expression, and returns this result as a float. Note:

You are guaranteed that the only operations present in the string are + and -.
Every number and operator in the string will be separated by one ' ' space character.
You will not be tested on any inputs that cannot be evaluated.
See the test cases for examples. Remember, this function should always return a float.
As usual (and regardless of whether the autograder stops you), you may not use concepts we have not yet covered in lecture, like lists. You also may not use the built-in function eval() or any similar built-in that effectively evaluates the string for you.

Here are some test cases:

```
assert(almostEqual(simpleEval('1 + 2 + 3'), 6.0))
assert(almostEqual(simpleEval('1 - 2 - 3'), -4.0))

# The test cases use almostEqual due to floating point inaccuracy
assert(almostEqual(simpleEval('0.3 + 0.3 + 0.3 - 0.9 + 1'), 1.0))

# Note that these expressions may begin with a -
assert(almostEqual(simpleEval('- 100 + 50.5 - 25'), -74.5))
assert(almostEqual(simpleEval('- 123.45 + 123.45 + 5 - 4 + 0.12'), 1.12))
```

Define simpleEval(s) here or on the following page

```
# Begin or continue your answer here:
```