15-112
Fall 2026 Exam 1
September 22, 2026

Name:

Andrew ID:

- You may not use any books, notes, or electronic devices during this exam.
- **Do not tear any pages out of the exam**. Doing so may invalidate your exam.
- **Write legibly**; you may not receive credit if the instructor cannot read your scribbles.
- Show your work on the exam to receive credit.
- If you write an answer on the back of any pages, make sure to make a note of this in the expected answer location.
- All code samples run without crashing. Assume any imports are already included as required.
- Do not use these post-exam 1 topics/constructs: lists, tuples, sets, maps/dictionaries, recursion, or classes/OOP.

Do not write anything in the table below.

Question	Points	Score
1	27	
2	10	
3	18	
4	20	
5	25	
Total:	100	

1. **Code Tracing**

(a) (10 points) Write the output for the following short code segments.

Code	Output
`print(517 % 100 + 517 // 100)`	

`a = "3"`
`b = "4"`
`print(a + b * 2)`	

`n = 0`
`for i in range(2, 11, 3):`
`  n += i % 4`
`print(n)`	

`x = 2.5` `print("15" + str(int(x)) + str(x * 2))`	
`s = "computation"` `print(s[2:5] + s[::4] + s[-1])`	

`def f(s):`
`  s = s + "!"`
`  return len(s)`
`s = "hi"`
`f(s)`
`print(s, len(s))`	

`x = "aaa"`
`y = "b"`
`while len(y) < len(x):`
`  y += y`
`print(y)`	

`s = "Exam One"`
`print(s.count("n"), s.find("m"),`
`  s.replace("n", "N"))`	

`print(2 + 2 * 2 ** 2 * 2)`	

`def p(x):`
`  print("p", x)`
`  return x % 3 == 0`
 	
`def q(x):`
`  print("q", x)`
`  return x > 5`
 	
`print(p(6) or q(4))`	

(b) (9 points) Indicate what the following program prints. Place your answers (and nothing else) in the box below the code.

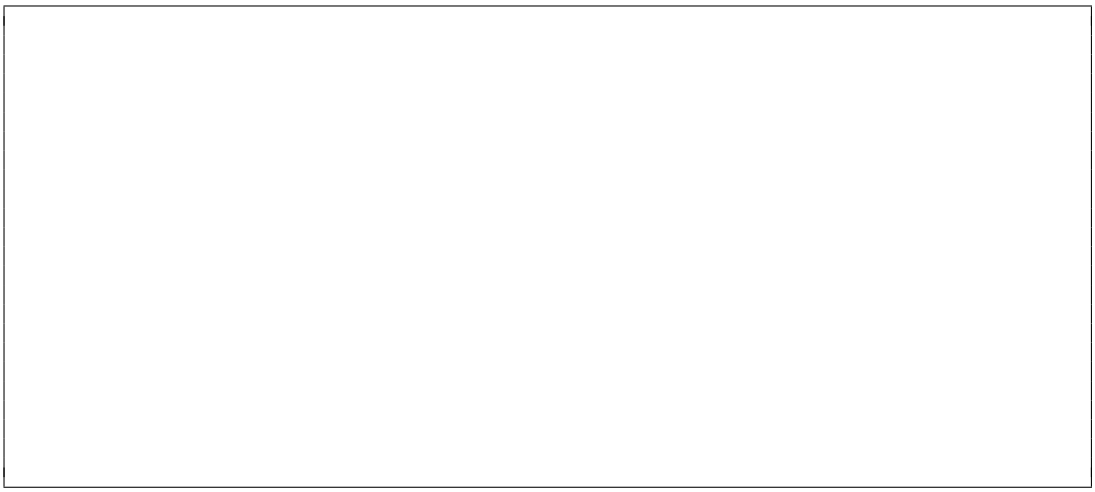
```
def ct2(n):
    total = 0
    for i in range(1, n):
        if n % i != 0:
            continue
        j = i
        while j < n:
            j += i
            total += 1
        print(i, j, total)
    return total

ct2(6)
```

(c) (8 points) Indicate what the following program prints. Place your answers (and nothing else) in the box below the code.

```
def ct3(s, t):
    r = ""
    for i in range(len(t)):
        k = s.find(t[i])
        if k < 0:
            r = t[i] + r
        else:
            r = r + s[k:k+2]
            s = s[:k] + s[k+1:]
        print(i, s, r)

print(ct3("banana", "and"))
```



2. (10 points) **Free Response:** Packing Boxes

   Do not use loops, strings, lists, tuples, dictionaries, the `math` *library, try/except, or recursion on this problem. If you do, you will receive a 0.*

   You are packing up your dorm room. Every box holds the same number of items, and a box may be left partly full.

   Write the function `boxesNeeded(items, perBox)` that takes two integers and returns the smallest number of boxes that will hold all `items` items, when each box holds at most `perBox` of them.

   If `items` is negative, or if `perBox` is zero or negative, the request does not make sense, so return `None`.

   Consider the following test cases:

```
assert boxesNeeded(10, 5) == 2     # exactly two full boxes
assert boxesNeeded(11, 5) == 3     # the 11th item needs a box of its own
assert boxesNeeded(6, 5) == 2
assert boxesNeeded(5, 5) == 1
assert boxesNeeded(1, 5) == 1
assert boxesNeeded(0, 5) == 0      # nothing to pack, so no boxes
assert boxesNeeded(7, 3) == 3
assert boxesNeeded(115, 10) == 12
assert boxesNeeded(5, 1) == 5

assert boxesNeeded(-4, 5) == None
assert boxesNeeded(10, 0) == None
assert boxesNeeded(10, -2) == None
```

3. **Free Response:** Switching Numbers

Do not use strings, lists, tuples, dictionaries, try/except, or recursion on this problem. If you do, you will receive a 0.

We will say that a positive integer is a **Switching** number if, reading its digits and alternating between two piles, the two piles have the same sum. That is, the sum of the 1st, 3rd, 5th, ... digits equals the sum of the 2nd, 4th, 6th, ... digits.

For example:

- 1034 is a Switching number: the first pile is $1 + 3 = 4$ and the second pile is $0 + 4 = 4$.
- 121 is a Switching number: the first pile is $1 + 1 = 2$ and the second pile is just 2.
- 209 is *not* a Switching number: $2 + 9 = 11$, but the other pile is only 0.
- For a longer example, 56847923 is a Switching number. Dealing its eight digits alternately into two piles:

$$\begin{array}{rl} \text{first pile} & 5 + 8 + 7 + 2 = 22 \\ \text{second pile} & 6 + 4 + 9 + 3 = 22 \end{array}$$

- No single-digit number is a Switching number, since one pile is empty (and so sums to 0) while the other is not.

The first 21 Switching numbers are 11, 22, 33, 44, 55, 66, 77, 88, 99, 110, 121, 132, 143, 154, 165, 176, 187, 198, 220, 231, 242.

(a) (13 points) Write the function `isSwitching(n)`, which takes a positive integer `n` and returns `True` if `n` is a Switching number and `False` otherwise. You may write any additional helper functions that you need.

```
assert isSwitching(1034) == True
assert isSwitching(121) == True
assert isSwitching(11) == True
assert isSwitching(220) == True
assert isSwitching(56847923) == True
assert isSwitching(56847924) == False   # one digit different
assert isSwitching(209) == False
assert isSwitching(1234) == False
assert isSwitching(100) == False
assert isSwitching(5) == False
```

Write your answer on the next page.

Answer Space for Question 3a

(b) (5 points) Write the function `nthSwitching(n)`, which takes a non-negative integer `n` and returns the nth Switching number. As is customary, we start counting at 0, so `nthSwitching(0)` returns 11.

You may assume that your implementation of `isSwitching(n)` works correctly, even if your own does not.

```
assert nthSwitching(0) == 11
assert nthSwitching(8) == 99
assert nthSwitching(9) == 110
assert nthSwitching(10) == 121
assert nthSwitching(18) == 220
```

4. (20 points) **Free Response:** X Marks the Spot

A square grid of characters is given to you as a single string, whose characters fill the grid one row at a time. If `len(s) == n*n`, then the grid has `n` rows and `n` columns. So the 9-character string `"X.X.X.X.X"` describes this grid:

```
X . X
. X .
X . X
```

We will say that such a grid is **marked** if both of its diagonals are drawn with the capital letter `X` — that is, if every cell on the top-left-to-bottom-right diagonal and every cell on the top-right-to-bottom-left diagonal holds an `X`, forming a large X across the grid. The grid above is marked. The characters that are not on a diagonal do not matter at all.

Write the function `xMarksTheSpot(s)` that takes a string `s` and returns `True` if it describes a marked grid, and `False` otherwise.

If `len(s)` is not a perfect square, the string does not describe a square grid at all, and you must return `False`. You must also return `False` for the empty string.

Consider the following test cases:

```
assert xMarksTheSpot("X.X.X.X.X") == True   # the 3x3 grid above
assert xMarksTheSpot("XXXXXXXXX") == True   # every cell is an X
assert xMarksTheSpot("XaaXaXXaaXXaXaaX") == True   # a 4x4 grid
assert xMarksTheSpot("XXXX") == True        # 2x2: all four cells are diagonal
assert xMarksTheSpot("X") == True           # 1x1

assert xMarksTheSpot("X.X.O.X.X") == False  # the center is on BOTH diagonals
assert xMarksTheSpot("XOXOXOOOX") == False  # the bottom-left corner is an O
assert xMarksTheSpot("XXOXOXOXX") == False  # six X's, but not on the diagonals
assert xMarksTheSpot("X..X") == False       # only one diagonal is drawn
assert xMarksTheSpot("aaaaaaaaa") == False  # a big A is not a big X
assert xMarksTheSpot("abcde") == False      # 5 is not a perfect square
assert xMarksTheSpot("") == False
```

Notes:

- *Both* diagonals must be drawn. `"X..X"` is the 2x2 grid with `X` in the top-left and bottom-right corners only, so it is not marked.
- When `n` is odd, the center cell lies on both diagonals.
- You do not know `n` in advance; it must be computed from `len(s)`.

Write your answer on the next page.

Answer Space for Question 4

5. **Free Response:** Compress and Decompress

   Reminder: No lists.

   A string of letters can be **compressed** by replacing each run of consecutive identical characters with the length of the run followed by that character. A run of length 1 is written as just the character, with no number in front of it.

   For example, `"aabbbc"` compresses to `"2a3bc"`: there are two `a`'s, then three `b`'s, then a single `c`.

   (a) (15 points) Write the function `compress(s)` which takes a string `s` containing only lowercase letters and returns the compressed version of it, as described above.

```
assert compress("aabbbc") == "2a3bc"
assert compress("abcd") == "abcd"          # no runs at all
assert compress("aabaa") == "2ab2a"        # runs need not be adjacent
assert compress("aaabbbaaa") == "3a3b3a"
assert compress("mississippi") == "mi2si2si2pi"
assert compress("zzzzzzzzzzzz") == "12z"   # counts can be 2 digits
assert compress("x") == "x"
assert compress("") == ""
```

   Note: you may assume `s` contains only lowercase letters, so you never have to worry about a digit appearing in the input.

Write your answer on the next page.

Answer Space for Question 5a

(b) (10 points) Write the function `decompress(s)` which takes a compressed string as produced by part (a) and returns the original string.

```
assert decompress("2a3bc") == "aabbbc"
assert decompress("abcd") == "abcd"
assert decompress("2ab2a") == "aabaa"
assert decompress("mi2si2si2pi") == "mississippi"
assert decompress("12z") == "zzzzzzzzzzzz"
assert decompress("100wq") == "w"*100 + "q"
assert decompress("") == ""
```

Notes:

- A number in the compressed string may have more than one digit, as in `"12z"` — that is twelve `z`'s, not one `z` followed by two more.
- You may assume `s` is a well-formed compressed string; you do not need to detect bad input.

References

Function	**Description**
`upper()`	Converts all characters to uppercase.
`lower()`	Converts all characters to lowercase.
`isalnum()`	Returns `True` if all characters are alphanumeric.
`isalpha()`	Returns `True` if all characters are alphabetic.
`isdigit()`	Returns `True` if all characters are digits.
`isspace()`	Returns `True` if all characters are whitespace.
`startswith(s)`	Returns `True` if string starts with `s`.
`endswith(s)`	Returns `True` if string ends with `s`.
`find(s)`	Returns the index of the first occurrence of `s`, or `-1` if not found.
`replace(old, new)`	Replaces all occurrences of `old` with `new`.
`split(sep)`	Splits string into a list by separator `sep`.
`splitlines()`	Splits string into a list at line breaks.
`strip()`	Removes leading and trailing whitespace.
`lstrip()`	Removes leading whitespace.
`rstrip()`	Removes trailing whitespace.
`format(...)`	Formats the string using placeholders.
`zfill(n)`	Pads the string with zeros to length `n`.
`capitalize()`	Capitalizes the first letter of the string.
`count(s)`	Returns the number of occurrences of `s`.

Table 1: Common Python String Functions