15-112
Spring 2025 Exam 1
February 18, 2025

Name:

Andrew ID:

- You may not use any books, notes, or electronic devices during this exam.
- You may not unstaple any pages.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- Write your answers in the specified places. If you run out of space for an answer, you may write on the backs of pages, but make sure to write a note telling the grader where to look for the rest of your answer.
- You may not use your own scrap paper. If you must use additional scrap paper, raise your hand, and we will provide some. You must hand any scrap paper in with your paper exam; we will not grade it.
- All code samples run without crashing. Assume any imports are already included as required.
- You may assume that cmu_graphics, math, string, and copy are imported; do not import any other modules.
- Do not use these post-midterm 1 topics/constructs: lists, sets, maps/dictionaries, recursion, or classes/OOP.

Do not write anything in the table below.

Question	Points	Score
1	32	
2	18	
3	20	
4	30	
Total:	100	

1. **Code Tracing**

(a) (8 points) Write the output for the following short code segments:

Code	Output
`x = 15112` `print(x//100)`	

`A = "42"`
`B = "18"`
`print(B+A)`	

`s = "123"`
`for i in range(3):`
`  s += s[i]`
`print(s)`	

`s = "mo"`
`t = "c" + s.replace('o','u') +  s[-1]`
`print(t)`	

`x = "code"`
`y = "coding"`
`while x[:2] == y[:2]:`
`  y = y[2:]`
`  x = x[2:]`
`print(y)`	

`s = "112"`
`for e in s.split("1"):`
`  print(e, end="#")`	

`n = 2025` `print(n%100 + n//100)`	

`def f(x):`
`  x = 3 * x - 2`
`  return x`
`x = 2`
`print(f(f(x)), f(x), x)`	

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

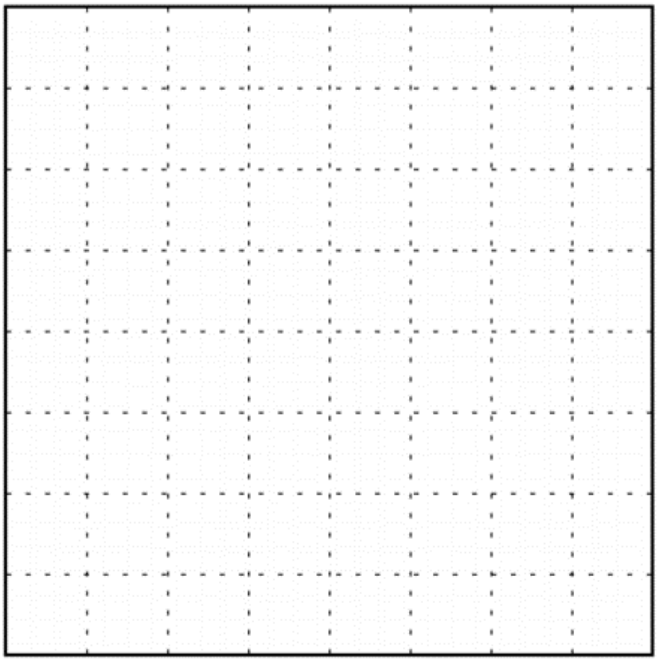
```
def ct1(s):
    t = s
    s = s + 'q'
    t = t + 'p'
    print(s + '=' + t)
    s = 's25'
    v = type(ord(s[0])) == float
    if type(v) == float:
        print("spring")
    else:
        print("break")
    t = str(int(s[1::2]) * 2)
    return t == 42

print(ct1('s25'))
```

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

```
def ct2(s):
    res = ""
    n = len(s)
    for i in range(5, 11):
        if i%2 == 1 and s[i%n].isdigit():
            res = res + s[i%n]
        elif i%2 == 0:
            res = s[i%n] + res
        else:
            continue
        if len(res) == 6:
            break
        print(res)
print(ct2("u1m2c1"))
```

(d) (8 points) Given that the box below is your canvas, with a width and height of 400 each, draw what the code below would display after precisely 5 seconds have passed since the start of the animation and the canvas has been updated.
Hint: Each of the small boxes on the canvas is 50x50 pixels.



```
from cmu_graphics import *

def onAppStart(app):
    app.cx = 50
    app.dx = 100
    app.counter = 0
    app.msg = ""
    app.stepsPerSecond = 30

def onStep(app):
    app.counter += 1
    if app.counter % 60 == 0:
        app.msg += "tac"
    elif app.counter % 30 == 0:
        app.msg += "tic"
        app.cx += app.dx
        if app.cx + 50 >= app.width:
            app.cx = app.width - 50
            app.dx *= -1

def redrawAll(app):
    drawLabel(app.msg, app.width//2, app.height - 30, size=30)
    drawCircle(app.cx, app.height//2, 50)

runApp(400,400)
```

2. **Free Response:** Spaced112 Numbers

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

   Note: You may not use strings in your solution for full credit. However, you will receive **half credit** *for a fully correct solution that uses strings*

   We'll say that an integer is a *Spaced112* number (coined term) if it is a positive integer such that...

   - It contains exactly three digits with values `1`, `1`, and `2`, appearing in that order within its digits.
   - These three digits occur one after the other, meaning that, reading from left to right, the first `1` appears before the second `1`, and the second `1` appears before the `2`, but they are **not all consecutive**.
   - The number does not contain any additional `1`s or `2`s beyond the three that form the `112` sub-sequence.

   For example, 1012 is a Spaced112 number because it contains the digits 1, 1, and 2 in that order, they are not consecutive, and there are no extra 1s or 2s. However, 21012 is not a Spaced112 number because it contains an extra 2. Similarly, 15112 is not a Spaced112 number because the digits 1, 1, and 2 appear next to each other without any separation.

   The first 10 Spaced112 numbers are 1012, 1102, 1132, 1142, 1152, 1162, 1172, 1182, 1192, 1312.

   (a) (10 points) Write the function `isSpaced112(n)`, which takes a positive integer `n` and returns `True` if `n` is a Spaced112 number and `False` otherwise. You can write any additional helper functions that you need.

Additional Space for Answer to Question 2a

(b) (8 points) Write the function `nthSpaced112(n)` which takes a non-negative integer `n` and returns the nth Spaced112 number. `nthSpaced112(0)` should return `1012`, the first Spaced112 number. You may assume that your implementation of `isSpaced112(n)` functions properly, even if yours does not.

3. (20 points) **Free Response:** Swappy Pairs

   Two strings `s1` and `s2` are Swappy Pairs if exactly one swap of two characters within `s1` transforms it into `s2`.

   Write a function `isSwappyPair(s1, s2)` that returns `True` if `s1` and `s2` are Swappy Pairs, meaning `s1` can be made identical to `s2` by swapping exactly two of its characters. Otherwise, return `False`.

   Here are some test cases:

```
# Basic swaps that should return True
assert isSwappyPair("hello", "hlelo")  # by swapping 'e' and 'l'
assert isSwappyPair("abcd", "abdc")  # by swapping 'c' and 'd'
assert isSwappyPair("racecar", "rrcecaa")
# by swapping the first 'a' and the last 'r'

assert isSwappyPair("abc", "abc") == False  # Identical strings
assert isSwappyPair("aaaa", "aaaa") == False  # Identical strings

assert isSwappyPair("abc", "xyz") == False  # Too many mismatches
assert isSwappyPair("abcde", "acbed") == False
# Two swaps required: 'b' 'c' and 'c' 'b'

assert isSwappyPair("apple", "appla") == False
# They cannot be made equal with any swap

assert isSwappyPair("abcd", "abcde") == False  # Length mismatch
```

Additional Space for Answer to Question 3

4. **Free Response:** Falling Zaggle

   (a) (10 points) **Graphics**

   Write the function `drawZaggleBoard(app, word, h)`, which takes a string `word` and a non-negative integer `h`, representing the vertical offset (in pixels) from the top of the window to the topmost squares.

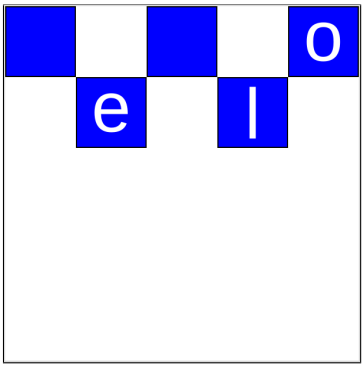
   As illustrated in the examples below, the function draws `n` blue squares in a zigzag pattern, where `n` is the length of the word. Each square represents a character in `word`:

   - Uppercase characters in `word` are displayed centered within their respective squares, in lowercase.
   - Lowercase characters in `word` are considered unguessed and are represented as empty blue squares.

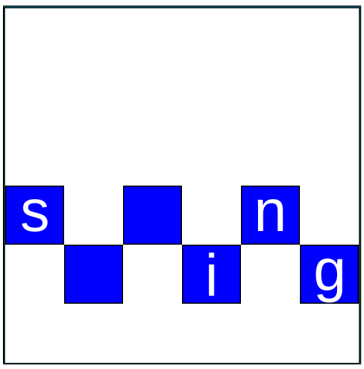
   The leftmost and rightmost squares should touch the left and right edges of the window, respectively. All elements must dynamically resize to adapt to window changes.

   Examples:

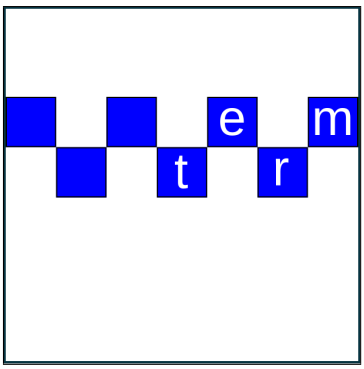
   `drawZaggleBoard(app, "hElLO", 0)` produces the graphics below:



   `drawZaggleBoard(app, "SprING", app.width//2)` produces the graphics below:



`drawZaggleBoard(app, "midTERM", app.width//4)` produces the graphics below:



Write the function `drawZaggleBoard(app, word, h)` below:

(b) (20 points) **The game**

Now, Let's animate the Falling Zaggle game.

In this part, assume the function `drawZaggleBoard(app, word, h)` already exists and works well. Do not write it again; instead, call it when you need it. Additionally, assume the existence of a helper function, `randomWord()`, which returns a random word. You do not need to implement it — use it when you need it.

The game should include the following features:

Game start:

- When the game begins, a random word is selected. An empty Zaggle board representation of that word appears at the top of the window.
- Initially, the blocks move downward at a speed of 5 pixels per second.

Character Guessing:

- The user can click inside a block to select it.
- Once selected, they can type a letter to guess the corresponding character.
- If the guess is correct, the letter appears inside the block.
- If the guess is incorrect, nothing happens, and the user must click inside a block again before making another attempt.

Game Over Condition:

- If the lowest blocks reach the bottom of the window before all characters are guessed, the game ends.
- The blocks disappear, and a "Game Over" message appears centered on the canvas.

Winning Condition:

- If the user correctly guesses all characters, they win the round.
- A new word is selected, the Zaggle board resets to the top, and the falling speed increases by 5 pixels per second (making the game progressively harder).

Game Reset

- Pressing the 'r' key resets the game.
- The speed returns to its initial value, and a new word is chosen.

Notes:

- Do not hardcode for a 400x400 canvas.
- You will be penalized if your code results in an MVC violation.
- Make reasonable choices for anything not specified above.
- To solve this, you need to write `onAppStart`, `onKeyPress`, `onMousePress`, `redrawAll`, and `onStep`.
- You do not need to include imports or the main function.

Additional Space for Answer to Question 4b

Additional Space for Answer to Question 4b

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

Animation functions:

CMU Graphics Reference: