

15-112 Fall 2021 Midterm 1A

80 minutes

Name: \_\_\_\_\_

Andrew ID: \_\_\_\_\_@andrew.cmu.edu

Section: \_\_\_\_\_

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam (not scratch paper) to receive credit.
- If you do use scratch paper, you must submit it with your andrew id on it, and we will ignore it.
- All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.
- Do not use recursion, sets, dictionaries, or imports we have not covered in class (e.g.. NumPy)
- You may use `almostEqual()` and `roundHalfUp()` without writing them. You must write everything else.
- Do not unstaple the exam

Do not write below here

| Question                | Points | Score |
|-------------------------|--------|-------|
| 1. Short Answer 1       | 4      |       |
| 2. Short Answer 2       | 4      |       |
| 3. Code Tracing 1       | 8      |       |
| 4. Code Tracing 2       | 8      |       |
| 5. makeDuplicateMatrix  | 16     |       |
| 6. firstNAcceptedValues | 30     |       |
| 7. Dots Animation       | 30     |       |
| 8. bonusCT 1 + 2        | 2      |       |
| <b>TOTAL</b>            | 100    |       |

**Short Answer 1 [4 pts]:** For each of the following possible directions in word search, write the appropriate values for (drow, dcol). Remember, (drow, dcol) is the direction the word is spelled from the first letter to the last letter, so the direction of "abc" in the list below would be considered "Right"

[ [ 'a', 'b', 'c' ],  
 [ 'd', 'e', 'f' ],  
 [ 'g', 'h', 'i' ] ]

1a. "Right" (drow, dcol) = \_\_\_\_\_

1b. "Down" (drow, dcol) = \_\_\_\_\_

1c. "Down-Left" (drow, dcol) = \_\_\_\_\_

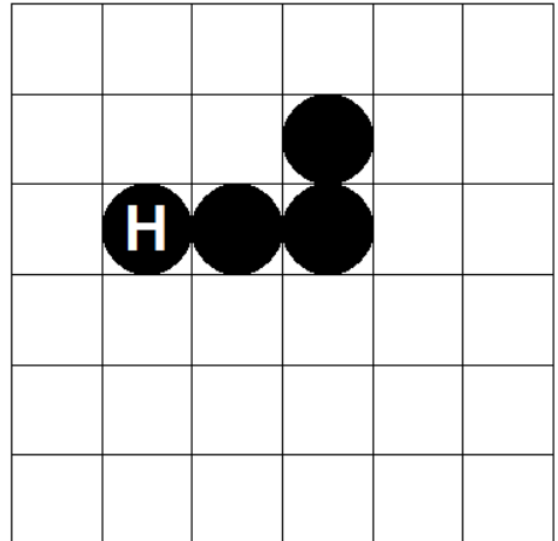
1d. "Down-Right" (drow, dcol) = \_\_\_\_\_

**Short Answer 2 [4 pts]:** Look at the picture of our Snake case study. In its current

state, write what we would see if we call `print(app.snake)`.

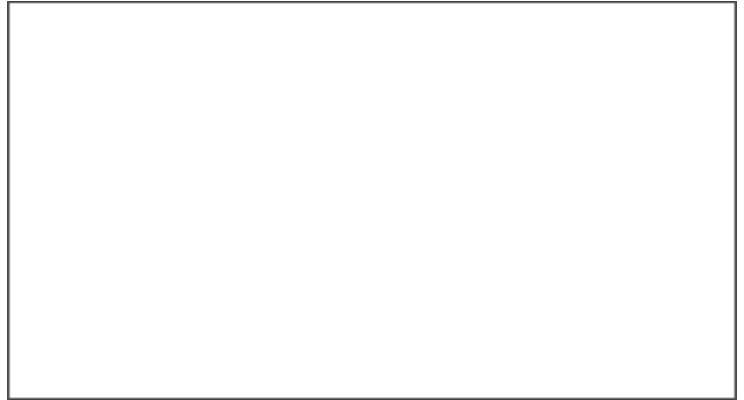
"H" is the head of the snake.

**Answer:**



**Code Tracing 1 [8pts]:** Indicate what the following code prints. Place your answers (and nothing else) in the box to the right of the code.

```
def ct1(s):
    r = s
    for i in range(len(s)):
        c = s[i]
        if c.islower():
            r += c
        elif c.isupper():
            r = c + r
        elif c.upper() == c:
            m = len(r)//2
            r = r[:m] + c + r[m:]
        print(f"{i} : {r}")
    return s
print(ct1('pH2'))
```



**Code Tracing 2 [8pts]:** Indicate what the following code prints. Place your answers (and nothing else) in the box to the right of the code.

```
import copy
def ct2(L):
    A = copy.deepcopy(L)
    B = [A[0], L[1]]
    B[1][0] = 'n'
    A[0].pop()
    A = A[1] + A[0]
    B[1] = sorted(B[0])
    print(f'A:{A}')
    print(f'B:{B}')
```



```
C = [[5, 2, 7], ['a', 'z']]
ct2(C)
print(f'C:{C}')
```

**Free Response 1: makeDuplicateMatrix [16pts]**

Write the function `makeDuplicateMatrix(n)` that takes an **even** positive integer  $n$ , and returns an  $n$ -by- $n$  2d list that contains all the integers from 1 up to (and including)  $(n^2)/2$  twice (so two 1's, two 2's, ...), all in random positions, so each time you call this function you get the same values but in different random positions. For example, `makeDuplicateMatrix(4)` could return this:

```
[[6, 8, 7, 4],  
 [5, 6, 3, 1],  
 [4, 8, 1, 2],  
 [5, 3, 2, 7]]
```

This is a 4-by-4 2d list with two 1's, two 2's, etc... and finally two 8's, all in random positions.

**Note:** You may **not** assume `make2dList()` has been provided.

**Hint:** You can randomly shuffle a 1d list  $L$  using `random.shuffle(L)`, which is destructive and returns `None`. This works on 1d lists, not 2d lists, so you'll have to adapt it somehow to solve this problem.

**Free Response 2: isAcceptedValue and firstNAcceptedValues [30pts]**

This problem includes a list of rules that either require or forbid certain positive integers. For example:

[ 'x must be a multiple of 3' ]

The first 6 integers that follow this rule are: [3, 6, 9, 12, 15, 18]

We can add a second rule to the list:

[ 'x must be a multiple of 3',  
'x must not be a multiple of 9' ]

The first 6 integers that follow these rules are: [3, 6, 12, 15, 21, 24]

We can add two more rules:

[ 'x must be a multiple of 3',  
'x must not be a multiple of 9',  
'x%2 must be a multiple of 2',  
'x%10 must not be equal to 4' ]

The first 6 integers that follow these rules are: [6, 12, 30, 42, 48, 60]

In general, the rules will have 3 parts, each with 2 options:

Part 1: 'x' or 'x%N' where N is a positive integer

Part 2: 'must be' or 'must not be'

Part 3: 'a multiple of M' or 'equal to M' where M is a non-negative integer.

We will only test your code on valid rules.

With this in mind, **first write the helper function isAcceptedValue(x, rules)**, which takes a positive integer x and a list of rules and returns True if x follows all the rules and False otherwise.

**Next, write the function firstNAcceptedValues(n, rules)** that takes a positive integer and a non-empty list of rules, and returns a list containing the first n positive integers that follow all the given rules.

Hint 1: Because the format is so restricted, you can check for just a few specific strings to determine which kind of command each line is. For example, to tell if we are using x or x%n, just check if '%' is anywhere in the line.

Hint 2: Some methods like .split() and .splitlines() return lists, which you now know how to index into. (But you do not necessarily need to use those methods.)

**Begin your answer on the next page!**

**(You may answer FR2 isAcceptedValue and firstNAcceptedValues on this page.)**

**(You may answer FR2 isAcceptedValue and firstNAcceptedValues on this page.)**



**Free Response 3: Dots animation [30pts]**

Important notes:

- For **full credit**, you need to solve this using `app.dots`, a list of instances of the **Dot dataclass** that you need to write. Also:
- You may abbreviate `app`, `event`, and `canvas` as `a`, `e`, and `c`.
- Assume reasonable values for anything not specified.

Using an appropriately-defined `Dot` dataclass, write an animation such that:

- The canvas is empty at the start except it displays "0 Dot(s)" at the top.
- Each time the user presses the mouse, a new dot is created, unless the press is inside an existing dot.
- New dots are created centered on the mouse press.
  - New dots are green, with a radius of 20.
  - The dot moves in a random direction while green. Each dot object should have its own `dx` and `dy` values (the change in `x` and `y` on each `timerFired`). You can use `random.randint(-3,3)` to get a new random value for each.
- Clicking on a dot changes the dot's color, from green to red or from red to green. (If a mouse click is inside two or more overlapping dots, they should each change.)
- Red dots do not move. Only green dots move.
- If the center of a dot goes off the canvas, the dot is deleted.
- The counter that started at "0 Dot(s)" updates to always reflect the current number of dots. Deleted dots are not counted.

Hint: to refresh your memory, here are a few lines from the course notes about dataclasses:

```
from dataclasses import make_dataclass
Dog = make_dataclass('Dog', ['name', 'age', 'breed'])
dog1 = Dog(name='Dino', age=10, breed='shepherd')
```

**Begin your answer on the next page!**

**(You may answer FR3: Dots Animation on this page.)**

**(You may answer FR3: Dots Animation on this page.)**

**Bonus/Optional Code Tracing 1 [1pts]**

Indicate what this prints. Place your answer (and nothing else) in the box.

```
import string
def bonusCt1(s):
    t = s
    result = 10**10
    n = 0
    for i in range(len(s)):
        s = s[::-1]
        n = 10*n + t.find(s[i])
        if (n > 12345):
            result = min(result, n)
            n = 0
    return result
```

```
print(bonusCt1(string.ascii_lowercase[:10]*2))
```

**Bonus/Optional Code Tracing 2 [1pts]**

Indicate what this prints. Place your answer (and nothing else) in the box.

```
def f(L):
    # helper fn for bonusCt2()
    R, C = len(L), len(L[0])
    M = [0] * (R + C - 1)
    for r in range(R):
        for c in range(C):
            M[r+c] = max(M[r+c], L[r][c])
    return M
```

```
def bonusCt2(L):
    return sum(f([[v, v-1, v-2, v-3] for v in f(L)]))[-2:]
```

```
print(bonusCt2([[1,2,3],
               [4,5,6],
               [7,8,9]]))
```