Name: _____

Please answer the following questions on scratch paper in which it is *clearly indicated what work corresponds to what problem.* Ensure that your work is easy to follow and that each problem is clearly labeled. When in doubt about something, write as much as you can about what you *do* know for the possibility of partial credit.

Each question clearly shows the number of points available and should serve as a rough metric to how much time you should expect to spend on each problem. In all questions, you may include or utilize functions or definitions that have been developed or used in the course by just supplying their name and a quick comment where they came from. Otherwise, comments are not required on the exam, but they may help me to better understand what you are trying to do and thus increase the chance at partial credit. Feel free to not include import statements, as I can figure out where things are supposed to be coming from.

The exam is partially open, and thus you are free to use:

- Your book
- Your notes
- Online slides and recorded lectures (if you think you have time)
- Any past work you have done for labs, problem sets, or projects

Beyond the above or submitting your work to Gradescope, you may not use a computer of any kind to assist with the test. Failure to abide by these rules will be considered a breach of Willamette's Honor Code and will result in penalties as set forth by Willamette's academic honesty policy.

Please sign and date the below lines to indicate that you have read and understand these instructions and agree to abide by them. *Failure to abide by the rules will result in a 0 on the test.* Good luck!!

_____ _____

Signature Date

Question:	1	2	3	4	Total
Points:	10	10	15	20	55
Score:					

(10) 1. **Evaluating Python Expressions:** For each of the below expressions, write out **both** the value of the expression and what type of object that value is (`int`, `float`, `bool`, etc). If the expression results in an error, state as much and indicate precisely where it went wrong. You can assume that the constant `HEXCHARS` has already been defined earlier in the code as `HEXCHARS = "0123456789ABCDEF"`, and has the internal representation shown below, where I've labeled both positive and negative indices.

HEXCHARS

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 -16	1 -15	2 -14	3 -13	4 -12	5 -11	6 -10	7 -9	8 -8	9 -7	10 -6	11 -5	12 -4	13 -3	14 -2	15 -1

(a)
```
8 * 5 + 7 % 4 - 9 // 3 ** 2 + 0 / 6
```

(b)
```
not (4 < 9) or str(5*2) != str(5)+str(2)
```

(c)
```
HEXCHARS[-5:] + HEXCHARS[-6::-3]
```

(d)
```
int(HEXCHARS[1])*HEXCHARS[-1]
```

(10) 2. **Program Tracing:** What output would the following program produce?

```
def puzzle(t):
    def mystery(r,x):
        x += 1
        def enigma(s):
            return r[s::x]
        return enigma
    x = 2
    y = mystery(t,x)
    return y(x) + y(0)

if __name__ == '__main__':
    print(puzzle("abcdefg"))
```

Show your work and explain your reasoning for full credit.

(15) 3. In many word puzzles or games, it is frequently desirable or useful to come up with new words which differ from a current word by only a single letter. For instance, if the starting word was `boat`, then words like `moat`, `boot`, and `boar` all would differ by only a single letter. The resulting word must be the same length as the original: imagine that you could just change a single letter to a different letter.

Here your task is to write a function called `one_off` that takes a single argument which will be the word as a string. Your function should then print out every word in the English language which differs from the input word by a single letter. At the end, it should *return* the number of words that were found. The below would be one example of some output:

```
>>> print(one_off("monkey"))
donkey
honkey
2
```

(20) 4. The game Lights Out was a small, electronic game in the late 90's in which the objective was to turn out all the lights on a small grid. Each time a grid cell was pressed, it would turn a light off if it was on *or* turn a light on if it was off, for both that cell *and its 4 adjacent neighbors*. The trick was to figure out which cells to press and in what sequence to get all the lights off at the same time. I was, and seeming still am, rubbish at it.

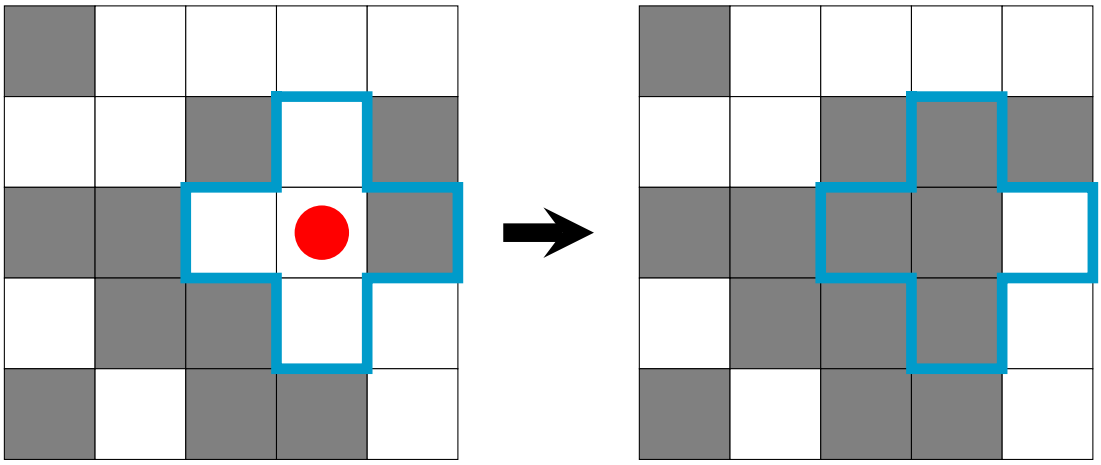

Figure 1: Clicking the cell indicated by the red circle would change the state of that cell and the 4 neighboring cells, flipping whether their lights were on or off.

Your task in this problem is to recreate this functionality, which turns out to be fairly straightforward in the PGL library. To get you started, I've defined a handful on constants below, to give you some easy values and numbers to work with. You do not need to rewrite these in your code, but please reference them and treat them as if they'd been defined above where your code started.

```
import random
from pgl import GWindow, GState, GRect

ROWS = 5                            # Number of rows in the grid
COLS = 5                            # Number of columns in the grid
CELL_SIZE = 80                      # Square size of each cell
GW_WIDTH = COLS * CELL_SIZE         # Width of entire window
GW_HEIGHT = ROWS * CELL_SIZE        # Height of entire window
ON_COLOR = "#FFFFFF"                # Color of cells which are "on"
OFF_COLOR = "#606060"               # Color of cells which are "off"

def lights_out():
    # and then your code would start!
```

You should ensure that your code achieves the following:

- Creates a grid of properly sized and placed cells on the window. Each cell should initially be randomly assigned an ON or OFF color (50/50 odds, one way or the other). Note in the example images above that the border color of all cells is still black, so that you can easily see the boundaries between cells.
- Whenever a user clicks a cell, it should toggle (ON becomes OFF, and OFF becomes ON) the fill color of that cell and the 4 adjacent cells.
- To spice things up a little, and to prevent impossible situations, every 20 seconds (20000 milliseconds), a cell should randomly be chosen and its light should be turned ON. If its light is already ON, nothing happens.
- For the sake of this exam, you do *not* need to check for any kind of victory state. Given my track record with this game, I'd never reach it anyway...

Remember that you are free to use any functions that you or the book have defined in the past, just leave me a quick comment explaining where they came from. If you have some false starts or go down some incorrect paths, just make sure that in the final paperwork you submit it is clear what you were doing (or attempting to do).