Name: ____________________

Please answer the following questions within the space provided on the online template. Format your solutions as well as you are able within the online text editor. While you are not required to document your code here, comments may help me to understand what you were trying to do and thus increase the likelihood of partial credit should something go wrong. If you get entirely stuck somewhere, explain in words as much as possible what you would try.

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. You can assume that you can import any of the common libraries we have used throughout the semester thus far.

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

- The text
- Your notes
- Class slides
- Any past work you have done as part of sections, problem sets, or projects, provided it has been uploaded, and you access it through GitHub.
- The in-Canvas calculator on problems that offer it.

While you are allowed to use a computer for ease of typing and accessing the above resources, you are prohibited from accessing and using any editor or terminal to run your code. Visual Studio Code or any similar editor should never be open on your system during this exam. Additionally, you are prohibited from accessing outside internet resources beyond the webpages described above. *Your work must be your own on this exam, and under no conditions should you discuss the exam or ask questions to anyone but myself.* 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

(10) 1. **Tracing Functions**

The code below defines three different function definitions:

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

    x = 2
    y = mystery(t,x=x)
    return y(x) + y()

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

For each function, every time that function would *return* a value, indicate what that value would be. If the function returns multiple times, put each value on a new line, in the order they would be returned.

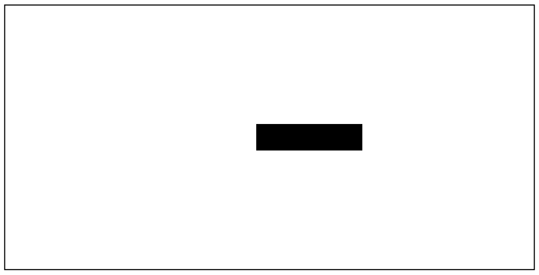
puzzle	mystery	enigma
`"gears"`	`enigma`	`"ge"`
		`"ars"`

Solution: The trickiest part of this is with `myster` returning the actual `enigma` function, which thus needs to track that the value of `x` within the surrounding `mystery` stack frame was 3.

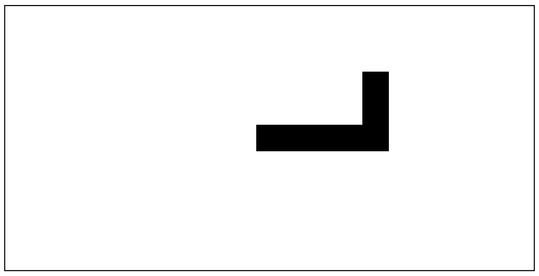
(20) 2. **Interactive Graphics**

In this problem, your mission is to write a program that plays a very simplified version of the graphical game called "Snake", which was popular on the first generation of Nokia phones. When the program begins, there is nothing on the graphics window, but there is an invisible snake head at the center of the window. In each time step of the animation, the snake head moves 15 pixels in some direction, leaving behind a 15×15 filled square at the position it just left.

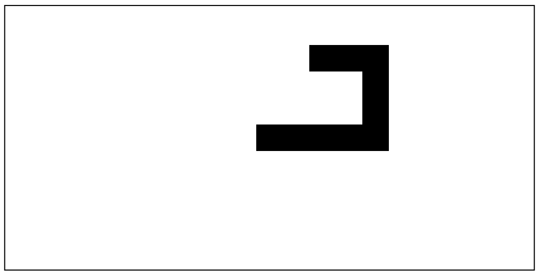
At the beginning of the game, the snake head is moving eastward, so after four time steps, it will have generated a trail of four squares (which run together on the screen), like this.



In this implementation of the game, you turn the snake by clicking the mouse. In this initial situation, with the snake moving horizontally, clicking the mouse above the current y position of the snake head sends it northward, and clicking below the current y position would send it south. In this particular example, let's assume that the player clicked above the snake head, so that its direction changes to the north. After three more time steps, the window would look something like this:



When the snake is moving vertically, clicking to the left of the snake head sends it westward, and clicking to the right sends it eastward. Clicking to the left would therefore send the snake off to the west, as follows:



In the actual game, the player loses if the path of the snake moves outside the window or closses its own path. For this problem, you can ignore the problem of stopping and just get the motion working.

While the program may seem complicated, there are not that many pieces which you need to get it working. Here are a few general hints or things to keep in mind as you get started:

- You need to keep track of three things: the x and y positions of the snake head, and some indication of what direction the snake is currently moving (North, South, East or West).
- The function that executes each time step must create a new black square whose center is at the current position, and *then* move the head on to the next square by updating the coordinates as appropriate to the current direction.
- The listener method that responds to mouse clicks has to look at the current position and direction and then use those together with the mouse click location to determine how to update the direction.
- Remember that you don't need to check whether the snake remains on the window or crosses its own path. You can implement those features on your own time after the exam if you really want!

Solution: One possible solution:

```
SIZE = 15
WIDTH = 20*SIZE
HEIGHT = 10*SIZE

def step():
    # Create rect at current position
    rect = GRect(gw.head_x - SIZE/2,
                 gw.head_y - SIZE/2,
                 SIZE, SIZE)
    rect.set_filled(True)
    gw.add(rect)
    # Advance head
    if gw.direction == 'E':
        gw.head_x += SIZE
    elif gw.direction == 'W':
        gw.head_x -= SIZE
    elif gw.direction == 'N':
        gw.head_y -= SIZE
    elif gw.direction == 'S':
        gw.head_y += SIZE
```

```
def click_action(e):
    mx = e.get_x()
    my = e.get_y()
    if gw.direction in 'NS':
        if mx < gw.head_x:
            gw.direction = 'W'
        else:
            gw.direction = 'E'
    else:
        if my < gw.head_y:
            gw.direction = 'N'
        else:
            gw.direction = 'S'

gw = GWindow(WIDTH, HEIGHT)
gw.head_x = WIDTH / 2
gw.head_y = HEIGHT / 2
gw.direction = 'E'
gw.set_interval(step, 200)
gw.add_event_listener("click", click_action)
```

(20) 3. **Working with Arrays**

Write a function called `rotate_array` that takes a list and an integer as two arguments. The function should have the effect of shifting every element of the list the integer number of positions. Positive integers should result in the elements being shifted toward the beginning of the list, whereas negative integers should result in the elements being shifted towards the end. Elements shifted off either end of the list should wrap around and reappear on the other end of the list. For example, if the array `digits` has the contents:

`digits`

0	1	2	3	4	5	6	7	8	9

then calling `rotate_array(digits, 1)` would shift each of the values one position to the left and move the first value to the end:

`digits`

1	2	3	4	5	6	7	8	9	0

Calling `rotate_array(digits, -3)` however would shift all the elements 3 positions to the right:

`digits`

7	8	9	0	1	2	3	4	5	6

Note that although the example array here just had integers as its elements, your function should properly shift *any* type of data the necessary number of positions. Also note that the function should shift the elements *in place*, it should not return a new list.

Solution: I found it useful here to figure out how to shift things 1 position in either direction, and then just use a loop to loop the necessary number of times. Other solutions certainly exist.

```python
def rotate_array(array, n):
    def roll_forward(array):
        tmp = array[0]
        for i in range(len(array)-1):
            array[i] = array[i+1]
        array[-1] = tmp
```

```python
def roll_backward(array):
    tmp = array[-1]
    for i in range(len(array)-1, 0, -1):
        array[i] = array[i - 1]
    array[0] = tmp

for i in range(abs(n)): # need for negatives
    if n > 0:
        roll_forward(array)
    else:
        roll_backward(array)
```