Practice Final 1

Name: ______________________________

Please answer the following questions within the space provided on the following pages. Should you need more space, you can use scratch paper, but clearly label on the scratch paper what problem it corresponds to. 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 printed portions of:

- The text
- Your notes
- Online slides
- Any past work you have done for labs, problem sets, or projects

Computers and internet capable devices are prohibited. *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

Question:	1	2	3	4	5	6	7	Total
Points:	10	10	15	20	15	10	20	100
Score:								

(10) 1. **Short Answer**

(a) Suppose that the function `f` is defined as follows:

```python
def f(x):
    def g(x):
        return len(
            [ i for i in range(1, x) if x % i == 0 ]
        ) == 1

    return sum([ i for i in range(x) if g(i) ])
```

What is the value of `f(10)`?

Solution: For a given digit, the list comprehension in `g` returns a list of all the integers up to, but not including that digit, which are even divisors. The output of `g` then checks if the length of that list is equal to 1, which is akin to asking if that number is prime, that is, if the number is divisible only by 1 (included in the list) and itself (not in the list). So `g` essentially returns `True` for prime numbers, and `False` for everything else.

Then we are just making a list of all the numbers up to `x=10` which are prime, and taking the sum of that list. Those prime numbers would be `[2,3,5,7]`, and thus they would sum to `17`.

(b) What value is printed if you call the function `example` in the following code?

```python
class MyClass:
    def __init__(self, x):
        def f(y):
            return 2 * x + y
        self.g = f

    def test(self, x):
        return self.g(x + 6)

def example():
    value = MyClass(14)
    print(value.test(8))
```

Solution: The function `f` is defined within the constructor, and thus has values defined within the constructor as part of its closure. Thus, when `test` is run the value of $8 + 6 = 14$ is passed in as `y` to `f`, resulting in a printed value of `2 * 14 + 14` or `42`.

(10) 2. **Simple Python** Write a Python program that reads integers that the user inputs on the console, ending when the user enters a blank line. At that point, the program should print out two lines, one showing the average of the odd numbers and one showing the average of the even numbers. Final decimals should always show two decimal places. An example run might look like:

```
〉 python prob2.py

Enter integers:
  ? 3
  ? 1
  ? 4
  ? 1
  ? 5
  ? 9
  ? 8
  ? 2
  ?
The average of the odd numbers is 3.80
The average of the even numbers is 4.67
```

The odd numbers in the input are 3, 1, 1, 5, and 9, which average to $(3+1+1+5+9)/5 = 3.8$, and the even numbers are 4, 8, and 2, which average to $(4+8+2)/3 = 4.666666$ and thus rounds to 4.67.

Solution:

```
odds = []
evens = []
finished = False
print("Enter integers:")
while not finished:
    num_str = input("  ? ")
    if num_str == "":
        finished = True
    else:
        num = int(num_str)
        if num % 2 == 0:
            evens.append(num)
        else:
            odds.append(num)
odd_avg = sum(odds)/len(odds)
```

```python
even_avg = sum(evens)/len(evens)
print(f"The average of the odd numbers is {odd_avg:.2f}")
print(f"The average of the even numbers is {even_avg:.2f}")
```

(15) 3. **Interactive Graphics**

The Portable Graphics Library fills a `GArc` by filling the wedge-shaped region formed by connecting the ends of the arc to the center, which turns out to be perfect for displaying a traditional pie chart. Your job in this problem is to write a function:

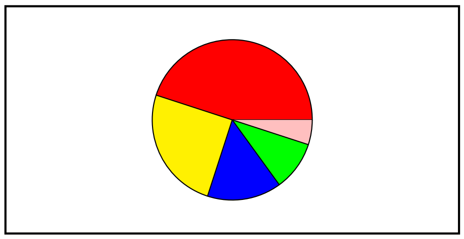
```python
def create_pie_chart(r, data):
```

that create a `GCompound` object for a pie chart with a set of data values, where `r` represents the radius of the circle, and `data` is the array of data values you wish to plot.

The operation of the `create_pie_chart` function is easiest to illustrate by example. If you execute the following test function:

```python
def test_pie_chart():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    data = [ 45, 25, 15, 10, 5 ]
    pie_chart = create_pie_chart(50, data)
    gw.add(pie_chart, gw.get_width()/2, gw.get_height()/2)
```

your program should generate the following pie chart in the center of the window:



The red wedge corresponds to the 45 in the data array and extends counterclockwise through 45% of the circle, which is not quite halfway. The yellow wedge then picks up where the left wedge left off and extends for 25% of a complete circle. The blue wedge takes up 15%, the green wedge takes up 10%, and the pink wedge the remaining 5%.

As you write your solution, keep the following points in mind:

- The values in the array are not necessarily percentages. What you need to do in your implementation is to divide each data cell by the sum of the elements to determine what fraction of the complete circle each value represents.
- The colors of the wedge are specified in the following constant array:

  ```python
  WEDGE_COLORS = [
      "Red", "Yellow", "Blue", "Green", "Pink", "Cyan"
  ]
  ```

  If you have more wedges than colors, you should just start the sequence over, so that the seventh wedge would be red, the eighth yellow, and so on.
- The reference point for the `GCompound` returned by `create_pie_chart` must be the center of the circle.

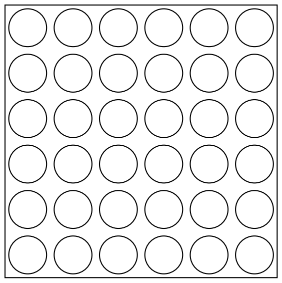
Solution: This is mostly about working with arcs and gcompounds:

```
from pgl import GWindow, GArc, GCompound

GWINDOW_WIDTH = 400
GWINDOW_HEIGHT = 200
WEDGE_COLORS = [
    "Red", "Yellow", "Blue", "Green", "Pink", "Cyan"
]

def create_pie_chart(r, data):
    comp = GCompound()
    total = sum(data)
    sweeps = [x/total*360 for x in data]
    ang_start = 0
    for i in range(len(data)):
        arc = GArc(-r, -r, 2*r, 2*r, ang_start, sweeps[i])
        arc.set_filled(True)
        arc.set_fill_color(WEDGE_COLORS[i % 6])
        comp.add(arc)
        ang_start += sweeps[i]
    return comp
```

(20) 4. **Interactive Graphics** One of the earliest electronic arcade games was *Whac-A-Mole*, which was released in 1976 by Creative Engineering, Inc. in Japan. In the game, the surface of the dispaly was covered with an arry of circular holes. From time to time, a "mole" would rise up out of the hole, and the player's job was to pound a hammer on that hole before the mole disappeared again below the surface. In this problem, your job is to create a simplified version of Whac-A-Mole, for which the implementation could be broken down into the following steps:

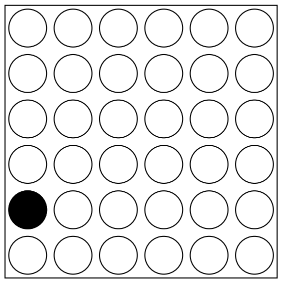
Step 1:

Write a program *whacamole.py* that displays a set of circles on the screen arranged to form a square matrix with `N_CIRCLES_PER_ROW` in each row and column. The diameter of each circle is given by the constant `CIRCLE_SIZE` and the space between each circle is given by the constant `CIRCLE_SEP`. You need to set the derived constants `GWINDOW_WIDTH` and `GWINDOW_HEIGHT` so that there is just enough room for the circles with a margin that is half the value of `CIRCLE_SEP` on all four sides. Thus, if `N_CIRCLES_PER_ROW` is 6, the initial display should look something like:



Step 2:

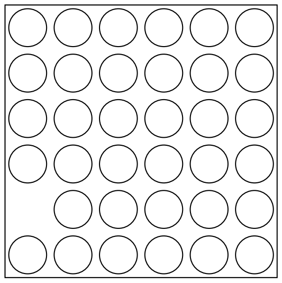
Set up a timer process that runs once every two seconds, where the time interval is defined as the constant `TIME_STEP`. In each time step, your program should pick a random (x, y) point somewhere in the graphics window. If this point is inside an unfilled circle, you should set the circle to be filled and set its fill color to `"Black"`, indicating the appearance of a mole. For example, if the random point is inside the circle just above the lower left circle, that circle should turn black, as follows:



Regardless of whether the chosen point is inside an unfilled circle, the code for each time step should change the previously selected circle to be unfilled, assuming that it still exists on the screen.

Step 3:

Set up a click handler that checks if the user clicks inside a circle during the time that it is filled (you can test whether a `GOval` is filled by calling the `is_filled` method). If the user manages to click in the circle before the timer process sets it back to its unfilled state, you should remove that circle from the graphics window. Thus, if the user clicks on the black circle in the preceding diagram within the `TIME_STEP` interval, that circle should disappear from the screen, as follows:



For the purposes of this problem, you need not figure out how to get the game to stop, which means that you can simply let the timer run until the user quits the program.

Solution: This may look a bit different from the solution in section, but this is how I approached it:

```
from pgl import GWindow, GOval
from random import randint

N_CIRCLES_PER_ROW = 6
CIRCLE_SIZE = 50
CIRCLE_SEP = 5

GWINDOW_WIDTH = (N_CIRCLES_PER_ROW *
                 (CIRCLE_SIZE + CIRCLE_SEP))
GWINDOW_HEIGHT = GWINDOW_WIDTH

def whacamole():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    for r in range(N_CIRCLES_PER_ROW):
        for c in range(N_CIRCLES_PER_ROW):
            x = (0.5 * CIRCLE_SEP +
                 c * (CIRCLE_SIZE + CIRCLE_SEP))
            y = (0.5 * CIRCLE_SEP +
                 r * (CIRCLE_SIZE + CIRCLE_SEP))
            oval = GOval(x, y, CIRCLE_SIZE, CIRCLE_SIZE)
            gw.add(oval)

    gw.prev = None
```

```python
def timer():
    x = randint(0, GWINDOW_WIDTH)
    y = randint(0, GWINDOW_HEIGHT)
    if gw.prev is not None:
        gw.prev.set_filled(False)
    target = gw.get_element_at(x,y)
    if target is not None:
        if not target.is_filled():
            target.set_filled(True)
            gw.prev = target

def click(e):
    x, y = e.get_x(), e.get_y()
    circ = gw.get_element_at(x,y)
    if circ is not None and circ.is_filled():
        gw.remove(circ)

gw.set_interval(timer, 700)
gw.add_event_listener("click", click)
```

(15) 5. **Strings** In Dan Brown's best-selling novel *The Da Vinci Code*, the first clue in a long chain of puzzles is a cryptic message left by the dying curator of the Louvre. Two lines of the message are

O, Draconian devil!
Oh, lame saint!

Professor Robert Langdon (the hero of the book, played by Tom Hanks in the movie) soon recognizes that these lines are ***anagrams***–pairs of strings that contain exactly the same letters–for

Leonardo da Vinci
The Mona Lisa

Your job in this problem is to write a predicate function `is_anagram` that takes two strings as arguments and returns `True` if they contain exactly the same alphabetic characters, even though those characters might appear in any order. Thus, your function should return `True` for each of the following calls:

```
is_anagram("O, Draconian devil!", "Leonardo da Vinci")
is_anagram("Oh, lame saint!", "The Mona Lisa")
is_anagram("ALGORITHMICALLY", "logarithmically")
is_anagram("Doctor Who", "Torchwood")
```

These examples illustrate two important requirements of the `is_anagram` function:

- The implementation should look only at letters, ignoring any extraneous spaces or punctuation marks that might show up along the way.
- The implementation should ignore the case of the letters in both strings.

Solution: There are a variety of ways this can be approached, but I found checking the letters individually to be difficult, as you'd need to be removing them as you went to ensure you got the correct counts. So I found comparing other structures directly to be more useful. My first method looked something like:

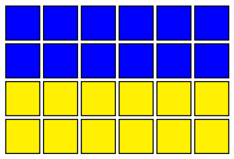
```
def is_anagram(s1, s2):
    s1 = s1.lower()
    s2 = s2.lower()
    return letter_counts(s1) == letter_counts(s2)

def letter_counts(s):
    counts = {}
    for letter in s:
        if letter.isalpha():
            counts[letter] = s.count(letter)
    return counts
```

and my second method like:

```
def is_anagram_v2(s1, s2):
    s1 = [s.lower() for s in s1 if s.isalpha()]
    s2 = [s.lower() for s in s2 if s.isalpha()]
    return sorted(s1) == sorted(s2)
```

(10) 6. **Working with Arrays** Write a function `double_image(old_image)` that takes an existing `GImage` and returns a new `GImage` that is twice as large in each dimension as the original. Each pixel in the old image should be mapped into the new image as a 2×2 square in the new image where each of the pixels in that square match the original.

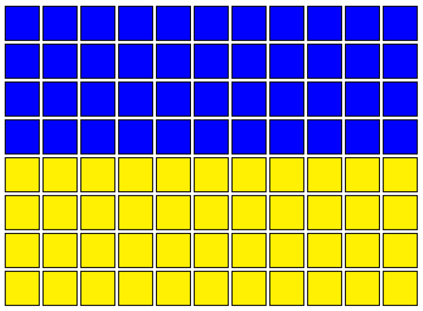
As an example, suppose that you have a `GImage` from the file `TinyUkrainFlag.png` that looks like the image below, where the scale has been expanded so that you can see the individual pixels, each of which appears as a small outlined square:



This 6×4 rectangle has two rows of blue pixels followed by two rows of yellow pixels. Calling:

```
bigger_ukrain_flag = double_image(GImage("TinyUkrainFlag.png"))
```

should create a new image with the following 12×8 pixel array:



The blue pixel in the upper left corner of the original has become a square of 4 blue pixels, the pixel to its right has become the next 2×2 square of blue pixels, and so on.

Keep in mind that your goal is to write an implementation of `double_image` that works with *any* `GImage`, and not just the flag image used in this example.

Solution: I probably could have done some sort of looping thing to "paint" all four new pixels in each case here, but I just figured with only 4 it would probably be more clear and about the same number of lines to just let each one explicitly.

```
from pgl import GWindow, GImage

def double_image(old_image):
    ow, oh = old_image.get_width(), old_image.get_height()
    new_image = [
            [0 for c in range(2 * ow)] for r in range(2 * oh)
        ]
        old_array = old_image.get_pixel_array()
        for r in range(oh):
            for c in range(ow):
                pixel = old_array[r][c]
                new_image[2 * r][2 * c] = pixel
                new_image[2 * r + 1][2 * c] = pixel
                new_image[2 * r][2 * c + 1] = pixel
                new_image[2 * r + 1][2 * c + 1] = pixel
        return GImage(new_image)
```

(20) 7. **Python Data Structures** In recent years, the globalization of the world economy has put increasing pressure on software developers to make their programs operate in a wide variety of languages. That process used to be called *internationalization*, but is now more often referred to (perhaps somewhat paradoxically) as *localization*. In particular, the menus and buttons that you use in a program should appear in a language that the user knows.

Your task in this problem is to write a definition for a class called `Localizer` designed to help with the localization process. The constructor for the class has the form:

```
class Localizer:
    def __init__(self, filename):
```

The constructor creates a new `Localizer` object and initializes it by reading the contents of the data file. The data file consists of an English word, followed by any number of lines of the form

```
xx=translation
```

where *xx* is a standardized two-letter language code, such as `de` for German, `es` for Spanish, and `fr` for French. Part of such a data file, therefore, might look like this:

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

This file tells us, for example, that the English word `Cancel` should be rendered in German as `Abbrechen`, in Spanish as `Ayudar`, and in French as `Annuler`.

Beyond the implementation of the constructor, the only method you need to define for `Localizer` is

```
def localize(self, word, language):
```

which returns the translation of the English word as specified by the two-letter language parameter. For example, if you have initialized a variable `my_localizer` by calling:

```
my_localizer = Localizer("Localizations.txt")
```

you could then call

```
my_localizer.localize("Open", "de")
```

and expect it to return the string "Öffnen". If no entry appears in the table for a particular word, `localize` should return the English word unchanged. Thus, `OK` becomes `Approuver` in French, but would remain as `OK` in Spanish or German.

As you write your answer to this problem, here are a few points to keep in mind:

- You can determine when a new entry starts in the data file by checking for a line without an equal sign. As long as an equal sign appears, what you have is a new translation for the most recent English word into a new language.
- For this problem, you don't have to worry about distinctions between uppercase and lowercase letters and may assume that the word passed to `localize` appears exactly as it does in the data file.
- The data file shown above is just a small example; your program must be general enough to work with a much larger file. You may not assume that there are only three languages or, worse yet, only four words.
- You don't have to do anything special for the characters in other languages that are not part of standard English, such as the ö and ß that appear in this data file. They are all characters in the expanded Unicode set that Python uses.
- It may help you solve this problem if you observe that it is the *combination* of an English word and a language code that has a unique translation. Thus, although there are several different translations of the word `Close` in the localizer and German translations of many words, there is only one entry for the combination of `Close+de`.

Solution: Taking the advice of the last bullet point, I decided to make my keys to the dictionary a tuple that held both the word and the language. You could have also do something similar by concatenating both together in a string or similar. Most of the rest of the complexity is just from reading in the file to the desired structure.

```
class Localizer:
    def __init__(self, filename):
        self._translations = self.read_file(filename)

    def read_file(self, filename):
        word = ""
        lang = ""
        trans = {}
        with open(filename) as fh:
            for line in fh:
                line = line.strip()
```

```python
                eqloc = line.find("=")
                if eqloc == -1:
                    word = line
                else:
                    lang = line[:eqloc]
                    new = line[eqloc + 1 :]
                    trans[(word, lang)] = new
        return trans

    def localize(self, word, lang):
        return self._translations.get((word, lang), word)
```