Practice Final 2

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. **Python Tracing**

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

```python
def conundrum():
    array = [ ]
    for i in range(6):
        array.append(0)
        for j in range(i, 0, -1):
            array[j] += j
    return array
```

Work through the function carefully and indicate the value of each of the elements of the array returned by a call to `conundrum`.

Solution: A new entry is appended each iteration, so there will be 6 values in the list. Each value starts at 0 and then has its index added to it $6 - \text{index}$ times. So index 0 has 0 added to it 6 times, index 1 has 1 added to it 5 times, index 2 has 2 added to it 4 times, etc. So the final list looks like:

```
[0, 5, 8, 9, 8, 5]
```

(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(10)
    print(value.test(-4))
```

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 $-4 + 6 = 2$ is passed in as `y` to `f`, resulting in a printed value of 22.

(10) 2. **Fundamental 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.

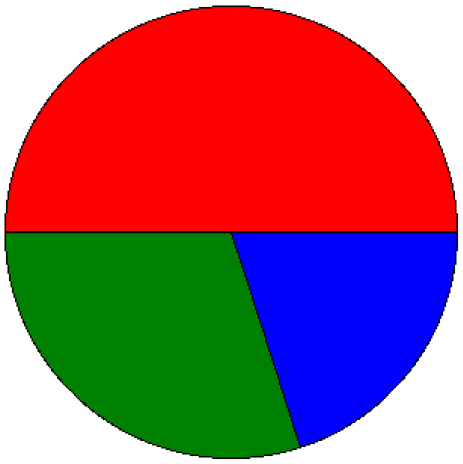
Solution:

```python
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)
```

```python
odd_avg = sum(odds)/len(odds)
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}")
```

(20) 3. **Interactive Graphics**

A pie chart is a common visualization method used to convey approximate relative percentages among several categories. Comprised of filled colorful wedges (which can be portrayed in this problem using a filled `GArc`), the idea is that each wedge has a size or takes up a portion of the overall circle equal to it's category percentage. For instance, the below image shows a pie chart with the red category taking up 50% of the circle, the green taking up 30% of the circle and then the blue taking up 20% of the circle.



Your task in this problem will be two-fold: you'd like to both create a pie-chart from a list of category percentages, but then you'd also like to add some simple mouse interactions.

Part 1

Define a function

```
def create_pie_chart(list_of_percents):
```

which will take one argument: a list of the various category percentages as integers. This list could have any number of elements, but you are guaranteed that all the integers within it will add up to 100. Your program should create the necessary `GWindow` and then add the needed filled `GArcs` to create the appearance of a pie chart centered in the window. You can imagine there are several constants already defined at the top of the file that you can use:

```
COLORS = ["red", "green", "blue", "orange"] #colors
GW_WIDTH = 500 # width of window
GW_HEIGHT = 500 # height of window
CHART_RADIUS = 150 # radius of pie chart
HIGHLIGHT_THICKNESS = 10 # line thickness when clicked
```

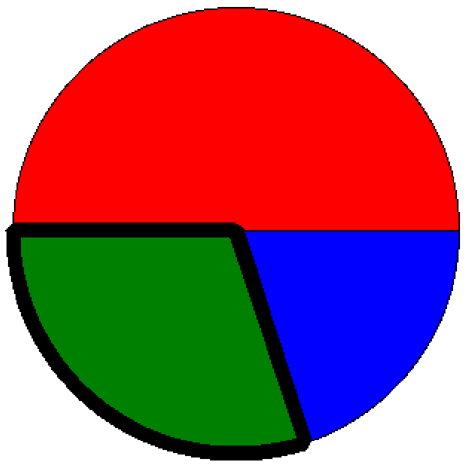
As you work through this part, a few extra things to keep in mind:

- The exact rotation of the overall pie chart doesn't matter, though it can be easiest to start the first wedge at `PGL`'s 0 angle.

- The desired colors of each wedge are given in the `COLORS` constant, and should be cycled through. If you have more wedges than colors, then the sequence should start over, so that the 5th wedge would also be red.
- The elements of the list are integers that represent a percentage, they are not angles. You will have to convert accordingly to what you need.

Part 2

We'd like to add some interaction to the pie chart, so that when one of the wedges is clicked, it is highlighted. To this end, you should add callback functions and corresponding listeners for both `"mousedown"` and `"mouseup"`. Upon pressing the mouse down on top of a wedge, the line width of that wedge should be increased to the specified `HIGHLIGHT_THICKNESS` and the wedge should be brought to the front of the stacking order (otherwise parts of the thick line remain behind other wedges, which looks weird). When the mouse is released, the line width should be reset back to 0 (you don't need to do anything to the stacking order). Pressing or releasing the mouse anywhere else on the screen should do nothing. An example of what the green wedge would look like while being clicked is shown below.



Solution:

```
from pgl import GWindow, GArc

COLORS = ["red", "green", "blue", "orange"]
GW_WIDTH = 500
GW_HEIGHT = 500
CHART_RADIUS = 150
HIGHLIGHT_THICKNESS = 10

def create_pie_chart(data):
    def down_action(event):
        elem = gw.get_element_at(event.get_x(),
                                 event.get_y())
        if elem is not None:
            elem.send_to_front()
            elem.set_line_width(HIGHLIGHT_THICKNESS)

    def up_action(event):
        elem = gw.get_element_at(event.get_x(),
                                 event.get_y())
        if elem is not None:
            elem.set_line_width(1)

    gw = GWindow(GW_WIDTH, GW_HEIGHT)
    start = 0
    i = 0
    for entry in data:
        stride = int(entry/100*360)
        x = GW_WIDTH / 2 -CHART_RADIUS
        y = GW_HEIGHT / 2 -CHART_RADIUS
        arc = GArc(x , y,
                   2*CHART_RADIUS,2*CHART_RADIUS,
                   start, stride)
        arc.set_filled(True)
        arc.set_fill_color(COLORS[i % len(COLORS)])
        gw.add(arc)
        start += stride
        i += 1

    gw.add_event_listener("mousedown", down_action)
    gw.add_event_listener("mouseup", up_action)
```

(15) 4. **Strings and Files**

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:

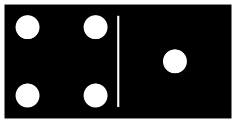
```
def is_anagram(s1, s2):
    s1 = s1.lower()
    s2 = s2.lower()
    return letter_counts(s1) == letter_counts(s2)

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) 5. **Class Definitions**

The game of dominoes is played using pieces that are often black rectangles with some number of white dots on each side. For example, the domino:



is called the 4-1 domino, with four dots on its left side and one on its right.

Define a `Domino` class that exports the following entries:

- A constructor that takes the number of dots on each side as integers
- A `__str__` method that creates a string representation of the domino, in whatever form you feel is best
- Two getter methods named `get_left_dots` and `get_right_dots` that return the number of dots on the left and right sides of the domino.
- One method called `will_match` which takes a single integer as an argument and returns `True` or `False` depending on whether either side of the domino would match that particular number.

You should be able to duplicate the following run with your code with similar results:

```
>>> d = Domino(4,1)
>>> print(d)
Domino(4,2)
>>> print(d.get_left_dots(), d.get_right_dots())
4 1
>>> print(d.will_match(4))
True
>>> print(d.will_match(6))
False
```

Solution: None of the individual methods is complicated here, it is more about knowing how to interact with `self` and work with classes in general.

```
class Domino:
    def __init__(self, left_dots, right_dots):
        self._left_dots = left_dots
        self._right_dots = right_dots

    def __str__(self):
        return f"Domino({self._left_dots},{self._right_dots})"
    def get_left_dots(self):
        return self._left_dots

    def get_right_dots(self):
        return self._right_dots

    def will_match(self, value):
        return (self._right_dots == value or
                self._left_dots == value)
```

(20) 6. **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 "Offnen". 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.

```python
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()
                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)
```