PROBLEM 1:  (Smith [Corona—Wesson] (32 points))

Part A (22 points)
Each of the variables below has a type and a value. The type is one of: list, boolean, int, string, float. For example, consider the assignment to variable x below:

\[ x = \text{len}([5,3,1]) \]

The type and value are shown in the first row of the table below. Fill in the other type and value entries based on the variable/expression in the first column.

<table>
<thead>
<tr>
<th>variable/expression</th>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ x = \text{len}([5,3,1]) ]</td>
<td>int or integer</td>
<td>3</td>
</tr>
<tr>
<td>[ a = 27/15 ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ h = 32 % 11 ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ e = \text{sum}([\text{range(6)}]) ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ d = \text{len}\text{&quot;dog&quot;} &lt; \text{len}\text{&quot;catsup&quot;} ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ c = \text{&quot;i,234,567&quot;.split\text{&quot;&quot;,&quot;&quot;}} ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ i = \text{&quot;platform&quot;[-2]} ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ g = \text{&quot;beat&quot; + &quot;nick&quot;} ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ b = \text{&quot;snapdragon&quot;[3:8]} ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ f = 0.23*10 ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ k = [8,7,6,7,6,5,4,3,2,1][5:7] ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ j = 2**4 ]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part B (4 points)
The Rohrer’s Index is an alternative to body mass index (BMI) as an anthropometric statistic. It is calculated using the formula below where weight is in pounds and height is in inches. Write the function `rohrer` whose header is provided below the formula. For example, `rohrer(165,73)` is \( \frac{165 \times 2768}{73^3} = 1.174 \)

\[
\text{rohrer}(\text{weight}, \text{height}) = \frac{\text{weight} \times 2768}{\text{height}^3}
\]

```python
def rohrer(weight, height):
    '''
    returns float value for rohrer’s index given
    int parameters weight and height
    '''
```

(continued)
Part C (6 points)

Heron's Formula for the area of a triangle with three sides $a, b, c$ can be calculated in two steps: the first calculates the semi-perimeter ($s$ below) and the second calculates the area ($A$ below):

\[ s = \frac{a + b + c}{2} \]
\[ A = \sqrt{(s-a) \times (s-b) \times (s-c) \times s} \]

For example, if the sides of a triangle are 3, 4, 5, then $s = (3 + 4 + 5)/2 = 6$ and the area is

\[ \sqrt{(6-3)(6-4)(6-5)(6)} = \sqrt{3 \times 2 \times 1 \times 6} = \sqrt{36} = 6 \]

Complete the function `triangle_area` that returns the area of a triangle with sides whose lengths are given by parameters `a`, `b`, and `c` (To compute a square root use the function `math.sqrt` or raise a number to the 0.5 power). For example, `triangle_area(3,4,5)` should evaluate to 6.0.

```python
def triangle_area(a, b, c):
    '''
    return area of triangle given int parameters a, b, c, lengths of sides
    '''
```

```
PROBLEM 2:  (Accumulated Wisdom (18 points))

Part A (5 points)
Write the function `divisor_sum` that returns the sum of the proper divisors of a number. For example:

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>divisor_sum(12)</td>
<td>13</td>
<td>$1 + 2 + 4 + 6 = 13$</td>
</tr>
<tr>
<td>divisor_sum(284)</td>
<td>220</td>
<td>$1 + 2 + 4 + 71 + 142 = 220$</td>
</tr>
<tr>
<td>divisor_sum(28)</td>
<td>28</td>
<td>$1 + 2 + 4 + 7 + 14 = 28$</td>
</tr>
</tbody>
</table>

def divisor_sum(n):
    return int, sum of proper divisor of n, n is an int > 0
    return 0
    return int, sum of proper divisor of n, n is an int > 0
    return 0
Part B (5 points)
A pair of numbers is *amicable* if the sum of the proper divisors of one number is equal to the other, and vice versa. For example, the pair (220, 284) is amicable because the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, and 110; the proper divisors of 284 are 1, 2, 4, 71, 142; and we have

\[
1 + 2 + 4 + 71 + 142 = 220
\]
\[
1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
\]

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_amicable(220, 284)</td>
<td>True</td>
</tr>
<tr>
<td>is_amicable(10, 35)</td>
<td>False</td>
</tr>
<tr>
<td>is_amicable(1184, 1210)</td>
<td>True</td>
</tr>
<tr>
<td>is_amicable(2214, 2688)</td>
<td>False</td>
</tr>
</tbody>
</table>

Write `is_amicable` below. **You should call `divisor_sum`, assume it works as specified.**

```python
def is_amicable(x, y):
    '''
    returns True if and only if int parameters x, y are amicable
    '''
```
Part C (8 points)

Cities or towns are specified by a three-element list such as 
['San Jose', 37.2406, -121.7457] where the first element is a string and the second two elements are float values specifying the latitude and longitude of the city.

Write the function `nearby` that has three parameters: the first parameter `city` is one three-element list specifying a city, the second parameter `clist` is a list of three-element lists specifying a list of cities, and the third parameter `apart` is a float representing a distance in miles. The function should return a list of strings: the names of the cities in `clist` that are no more than `apart` miles from `city`. For example, the call below would return the list ['Palo Alto', "Big Sur"], the only cities in `clist` within 150 miles of San Jose.

    nearby(["San Jose", 37.2406, -121.7457],
           [["Palo Alto", 37.2833, -121.9179],
            ["Princeton", 40.3436, -74.694],
            ["Big Sur", 35.9348, -121.46894], ["Los Angeles",34.0522, -118.2428]],
           150)

In writing `nearby` you must call the function `distance` shown below that takes two coordinate pairs of latitudes and longitudes and correctly returns the distance in miles between the coordinates.

    def distance(lat1, lon1, lat2, lon2):
        #
        # return distance in miles between (lat1,lon1) and (lat2,lon2), latitude and longitude pairs
        #
        x = 69.1*(lat1-lat2)
        y = 53.0*(lon1-lon2)
        return math.sqrt(x*x + y*y)

    def nearby(city, clist, apart):

PROBLEM 3:  (Big Ugly Gigantic Spiders (16 points))

Part A (4 points)
The function censored below is supposed to return "censored" if the string parameter headline contains any of the words in parameter bad, a list of strings, and return "clean" if headline does not contain any words in bad.

The table illustrates what censored is supposed to return.

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>censored([&quot;red&quot;,&quot;blue&quot;,&quot;green&quot;], &quot;red tide awesome&quot;)</td>
<td>&quot;censored&quot;</td>
</tr>
<tr>
<td>censored([&quot;red&quot;,&quot;blue&quot;,&quot;green&quot;], &quot;cardinal rules&quot;)</td>
<td>&quot;clean&quot;</td>
</tr>
<tr>
<td>censored([&quot;big&quot;,&quot;bad&quot;,&quot;trouble&quot;], &quot;bad news is trouble&quot;)</td>
<td>&quot;censored&quot;</td>
</tr>
<tr>
<td>censored([&quot;big&quot;,&quot;bad&quot;,&quot;trouble&quot;], &quot;bigfoot badboy troublesome&quot;)</td>
<td>&quot;clean&quot;</td>
</tr>
</tbody>
</table>

The implementation below is not correct. However it passes three of the four tests/examples shown above. Explain which one test the implementation below fails and why it fails.

```python
def censored(bad,headline):
    words = headline.split()
    for b in bad:
        if b in words:
            return "censored"
    else:
        return "clean"
```

Part B (4 points)
Write a correct version of censored below – this should work for all parameters, not just those shown in the table above. You can modify the code above, or copy/change it below.
The *Reverse Name* APT is attached at the end of this test. The function `change` is intended to reverse the last and first names when in the format *last, first*, so that the call below returns "bob jones".

```python
change("jones, bob")
```

Here’s one student’s solution that is all green. You’ll be asked two questions about this code.

```python
def change(name):
    index = name.find","
    return name[index+2:]+" "+name[:index]
```

**Part C (4 points)**

Explain in words why the first slice used in the return uses `index+2` and why the second slice uses `index`.

**Part D (4 points)**

Pat looks at the code and says it will generate an error message if it’s called with a string without commas (not allowed in the APT) so that `change("bob jones")` will generate an error. Ryan says no, it won’t generate an error, runs the code, and the call `change("bob jones")` returns the string below (no error is generated).

```
ob jones bob jone
```

Explain why the function generates this return value and does not result in an error.
PROBLEM 4:  \textit{(How many teens? How many boomers? (16 points))}

A. (8 pts) Write the function \texttt{getAges} which has one parameter \texttt{data} that is a nonempty list of strings in the format 'firstname:lastname:age' and returns a list of ints of the ages from \texttt{data}.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{call} & \textbf{returns} \\
\hline
\texttt{getAges(["Barack:Obama:50"]} & [50] \\
\hline
\end{tabular}
\end{center}

\begin{verbatim}
def getAges(data):
    \end{verbatim}

ASSUME PART A works. You will write Part A in lab.

B. (8 points) Write the function \texttt{howManyInRange} which has three parameters, \texttt{data} that is a list of strings in the format 'firstname:lastname:age', and two int parameters \texttt{start} and \texttt{end}. This function returns the number of people in the age range from start to end including the start and end ages. In writing \texttt{howManyInRange} you may call \texttt{getAges} that you wrote in Part A. Assume \texttt{getAges} works correctly.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{call} & \textbf{returns} \\
\hline
\texttt{howManyInRange(["A:A:8", "B:B:3", "C:C:17", "D:D:42", "E:E:20"],20,29)} & 1 \\
\texttt{howManyInRange(["Barack:Obama:50"],30,39)} & 0 \\
\hline
\end{tabular}
\end{center}

\begin{verbatim}
def howManyInRange(values, start, end):
    \end{verbatim}

PROBLEM 5:  \textit{(Talk like a Pirate (14 points))}

There are three simple rules for talking like a pirate.

1. The word 'Hello' (capitalized or not) becomes 'Ahoy' (always capitalized)
2. 'ar' not starting a word becomes 'arrr' (replace only the first occurrence)
3. For any word of length greater than 7 that does not contain 'ar' inside the word, remove all occurrences of lowercase o’s and u’s

Write the function \texttt{convertWord} that takes a word and returns the pirate equivalent of that word following the rules above.

\begin{verbatim}
def convertWord(word):
    \end{verbatim}
<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>convertWord('yesterday')</td>
<td>'yesterday'</td>
<td>no changes</td>
</tr>
<tr>
<td>convertWord('boing')</td>
<td>'boing'</td>
<td>word too short, no o replaced</td>
</tr>
<tr>
<td>convertWord('are')</td>
<td>'are'</td>
<td>'ar' starts a word, no change</td>
</tr>
<tr>
<td>convertWord('gargargantuan')</td>
<td>'garrrgargantuan'</td>
<td>only first 'ar' replaced</td>
</tr>
<tr>
<td>convertWord('purposefully')</td>
<td>'prpseffly'</td>
<td>o's and u's removed</td>
</tr>
<tr>
<td>convertWord('starboard')</td>
<td>'starrrboard'</td>
<td>no 'o' removed since 'ar' in word</td>
</tr>
</tbody>
</table>
Problem 1: (Play that Funky Music)

Part A
A number is perfect if it is the sum of its proper divisors, that is its divisors other than itself. The numbers 6, 28, and 496 are perfect as shown below:

\[
\begin{align*}
1 + 2 + 3 &= 6 \\
1 + 2 + 4 + 7 + 14 &= 28 \\
1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248 &= 496
\end{align*}
\]

Write the function `isPerfect` to return True if its parameter is perfect and False otherwise.

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPerfect(4)</td>
<td>False</td>
</tr>
<tr>
<td>isPerfect(6)</td>
<td>True</td>
</tr>
<tr>
<td>isPerfect(27)</td>
<td>False</td>
</tr>
<tr>
<td>isPerfect(28)</td>
<td>True</td>
</tr>
</tbody>
</table>

def isPerfect(num):
    
    return True if int parameter num is perfect and
    returns False otherwise

"""
Part B

Sometimes the beginning of each word in a list of words combine to make something like an acronym for the list. For example, if we take the first part of each word up to the first vowel and including one letter after the first vowel in each word of a group of words we can form a word from a phrase. For example the phrase probe lemming atrophy icing leads to prob + lem + at + ic = "problematic".

Write the function combine that creates a string from each of the strings in parameter phrase, a list of strings. In writing combine you can call the function firstVowelIndex given below that returns the index of the first vowel in a string. Every string in the list phrase will contain a vowel and the first vowel in each string will not be the last character of the string.

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>combine([&quot;money&quot;, &quot;itch&quot;, &quot;orange&quot;]))</td>
<td>&quot;monitor&quot;</td>
</tr>
<tr>
<td>combine([&quot;creamery&quot;, &quot;session&quot;]))</td>
<td>&quot;creases&quot;</td>
</tr>
</tbody>
</table>

```python
def firstVowelIndex(word):
    vow = "aeiou"
    for i, ch in enumerate(word):
        if vow.find(ch) >= 0:
            return i
    return len(word)

def combine(phrase):
    """
    return string formed by combining strings in the list phrase, each string contributes at least two letters, from the beginning of the string up to and including one letter after the first vowel.
    """
```

```python
```
PROBLEM 7:

PROBLEM 2**  (Phunkadelic (12 points))

Part A
A number is *square free* if it is not divisible by any perfect square greater than one. For example, 10 is square free since it is not divisible by four nor by nine, the two perfect squares less than 10. The number 100 is **not square free** since it is divisible by 25, and 25 is a perfect square and by 4 which is also a perfect square. Write the function `isSquareFree` to return True if its int parameter is square free, and false otherwise. For example:

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isSquareFree(8)</code></td>
<td>False</td>
</tr>
<tr>
<td><code>isSquareFree(45)</code></td>
<td>False</td>
</tr>
<tr>
<td><code>isSquareFree(38)</code></td>
<td>True</td>
</tr>
<tr>
<td><code>isSquareFree(55)</code></td>
<td>True</td>
</tr>
</tbody>
</table>

Hint: if you loop over 1,2,3,4, ... you can test divisors 1,4,9,16, ... by squaring each of the 1,2,3,4 being looped over.

```python
def isSquareFree(num):
    
    return True if int parameter num is square free and
    returns False otherwise
    ````
Part B
Some words contain other words. For example, each of "sublime", "compliment", "limerick" and "millimeter" contains the word "lime". Write the function `wordCount` that returns the number of strings in its list parameter `words` that contain the string `sub`.

```
def wordCount(words, sub):
    return the number of strings in string list words
    that contain string sub
```

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordCount(['sublime', 'millimeter', 'lemon'], 'lime')</td>
<td>2</td>
</tr>
<tr>
<td>wordCount(['subtract', 'assume', 'consumer', 'presume', 'lime'], 'sum')</td>
<td>3</td>
</tr>
<tr>
<td>wordCount(['apple', 'banana', 'lemon'], 'meat')</td>
<td>0</td>
</tr>
</tbody>
</table>
Part A (15 points)
The spelling idiom of *i before e except after c* can be checked by the Python function `spellwell` below (comments label each line of code).

```python
def spellwell(word):
    iei = word.find("ie")  #1
    eii = word.find("ei")  #2
    if iei == -1 and eii == -1: #3
        return True  #4
    if iei != 0 and word[iei-1] == 'c': #5
        return False  #6
    if eii != 0 and word[eii-1] == 'c': #7
        return True  #8
    return False  #9
```

For example, consider this run and the corresponding output:

```python
words = ["niece","either","receive","recieve","neighbor","piece","peice","dog"]
for x in words:
    print x, spellwell(x)
nice True
either False
receive True
recieve False
neighbor False
piece True
peice False
dog True
```

• One of the words shown above is labeled in the output as **True** by the return statement on line 4. Which word and why?

• Which line returns **True** for *receive* and why?

• Which line returns **False** for *recieve* and why?

* Why is the value false returned for "either"?
* No words in the English language start with "ie" or "ei", so "ei" at the beginning of a word should not be considered a violation of this spelling rule. Modify the code to reject any word that starts with "ie" and accept any word that starts with "ei".

Don't worry about multiple occurrences of "ei" or "ie" in a word.