hw4

Reading

Chapters 4 and 5.

Programming Problems

`vowelIndex`

Write a function `vowelIndex` that accepts a word (a `str`) as an argument and returns the **index** of the *first* vowel in the word. If there is no vowel, `-1` is returned. For this problem, both upper and lower case vowels count, and 'y' is **not** considered to be a vowel. Sample usage:

```
>>> vowelIndex('hello')
1
>>> vowelIndex('string')
3
>>> vowelIndex('strIng')
3
>>> vowelIndex('BUY')
1
>>> vowelIndex('APPLE')
0
>>> vowelIndex('nymphly')
-1
>>> vowelIndex('string')==3
True
>>> vowelIndex('nymphly')==-1
True
```

`flipCase`

Write a function `flipCase` that accepts a word as an argument and returns the same word but with each upper case letter switched to lower case and vice-versa. Sample usage:

```
>>> flipCase('ostrich')
'OSTRICH'
>>> flipCase('LyNx')
'lYnX'
>>> flipCase('Orangutan')
'oRANGUTAN'
>>> flipCase('AardVark')
'aARDvARK'
>>> flipCase('CamelCase')
'cAMELcASE'
>>> flipCase('ostrich')=='OSTRICH'
True
```

`palindromes`

Write a function `palindromes` that accepts a sentence as an argument. The function then returns a list of all *words* in the sentence that are palindromes, that is they are the same forwards and backwards. Guidelines:

- punctuation characters `.,;!?` should be ignored
- the palindrome check should *not* depend on case

Sample usage:

```
>>> palindromes("Hey Anna, would you prefer to ride in a kayak or a racecar?")
['Anna', 'a', 'kayak', 'a', 'racecar']
>>> palindromes("Able was I ere I saw Elba.")
['I', 'ere', 'I']
>>> palindromes("Otto, go see Tacocat at the Civic Center, their guitar solos are
wow!")
['Otto', 'Tacocat', 'Civic', 'solos', 'wow']
>>> palindromes("Otto, go see Tacocat at the Civic Center, their guitar solos are
wow!")==['Otto', 'Tacocat', 'Civic', 'solos', 'wow']
True
```

`squares`

Write a function `squares` that accepts a *2-dimensional list* of integers *or* a list of ranges as an input, and that *returns* the **count** of all the integers that are perfect squares.

- a perfect square is an integer that is the square of another integer. 9 is a perfect square because 9 is equal to 3 squared.
- don't worry about the ranges, if you write your code in the obvious way, it will also work for ranges.

Sample usage:

```
>>> squares( [[1,2,3],[4,5],[6,7,8,9]]) # 1,4,9 are perfect squares
3
>>> squares( [[1,2,3],[4,5,6],[7,8],[9,10,11,12],[13,14,15,16]] )
4
>>> squares( [ range(1,1000,7), range(1,500,13)])
12
>>> squares( [range(2,1000,3), range(7,100,8), range(8,1000,5)])
0
>>> squares( [ range(1,1000,7), range(1,500,13)])==12
True
```

`rps`

(Exercise 5.26 from the textbook) Write a function `rps` that returns the result of a game of "Rock, Paper, Scissors". The function accepts two arguments, each one of `'R','P','S'`, that represents the symbol played by each of the two players. The function returns:

- -1 if the first player wins
- 0 if a tie
- 1 if the second player wins
- Scissors beats Paper beats Rock beats Scissors

Sample usage:

```
>>> rps('R','P') # player 2 wins, return 1
1
>>> rps('R','S') # player 1 wins, return -1
-1
>>> rps('S','S') # tie, return 0
0
>>> [ (p1,p2,rps(p1,p2)) for p1 in 'RPS' for p2 in 'RPS']
[('R', 'R', 0), ('R', 'P', 1), ('R', 'S', -1), ('P', 'R', -1), ('P', 'P', 0),
('P', 'S', 1), ('S', 'R', 1), ('S', 'P', -1), ('S', 'S', 0)]
```
