**Chapter 5 test problems**

1. Suppose there is already a variable called time, containing an integer representing the current time. Write code that prints “Good morning!” if time is less than 1200; prints “Good afternoon!” if time is greater or equal to 1200 but less than 1800; and prints “Good evening!” otherwise.

2. Write function season() that takes as input the number of a month (1-12) and returns the season corresponding to the month, as a string. You may assume that numbers 1 to 3 correspond to Winter, 4 to 6 correspond to Spring, 7 to 9 correspond to Summer, and 10 to 12 correspond to Fall.

>>> season(3)

'Winter'

>>> season(11)

'Fall'

3. Write a function called qsolver() that solves a quadratic equation: ax2+bx+c=0. Your function will take three input arguments, a, b, and c, first compute the discriminant d = b2 - 4\*a\*c. If d < 0, the equation has no real roots. If d == 0 then the equation has one real root equal to -b / (2\*a). If d > 0, the equation has two roots: (-b - math.sqrt(d)) / (2\*a) and (-b+math.sqrt(d))/(2\*a). Your program should solve the equation given by coefficients a, b, and c, and then return string "No roots", or value \_\_\_\_ , or tuple (\_\_\_\_, \_\_\_\_).

>>> qsolver(1,2,1)  
-1.0  
>>> qsolver(1,0,-1)  
(1.0, -1.0)  
>>> qsolver(1,0,1)  
'No roots'

4. If there is a vote at a meeting, there are several possible outcomes based on the number of yes and no votes (abstains are not counted). If all the votes are yes, then the proposal passes "unanimously", if at least 2/3 of the votes are yes, then the proposal passes with "absolute majority", if at least 1/2 of the votes are yes, then the proposal passes by "simple majority", and otherwise it fails. Write function vote() that takes as input the number of yes votes and the number of no votes and then prints the outcome of the vote.

>>> votes(9, 0)

proposal passes unanimously

>>> votes(6, 3)

proposal passes with absolute majority

>>> votes(5, 4)

proposal passes with simple majority

>>> votes(4, 5)

proposal fails

5. Here are (slightly fictional) tax brackets:

|  |  |
| --- | --- |
| Taxable Income | Tax on this income |
| $0 – $6,000 | Nil |
| $6,001 – $37,000 | 15 cents for each $1 over $6,000 |
| $37,001 – $80,000 | $4,650 plus 30 cents for each $1 over $37,000 |
| Over $80,001 | $17,550 plus 37 cents for each $1 over $80,000 |

For example, suppose you make $90,000; then you fall into the 4th tax bracket which means you owe $17550 + $.37 \* (90000-80000) = $21250.00 in taxes. Write function tax() that as an input takes your (taxable) income and then calculates and returns how much tax you owe.

>>> tax(5500)

0

>>> tax(7000)

150

>>> tax(38000)

4950

>>> tax(81000)

17920

6. Write a loop that will repeatedly request whole numbers from the user. When the user enters 0, a report should be printed saying how many numbers were entered and their sum. The final 0 should not be counted as one of the numbers.

>>>

Enter a number: 5

Enter a number: -1

Enter a number: 0

There were 2 numbers entered, totaling 4

7. Write a function crypto() that takes as an input a string s and returns an encrypted string where encryption proceeds as follows: split the text up into blocks of two letters each and swap each pair of letters (where spaces/punctuation, etc. is treated like letters). If the input string

>>> crypto('Secret Message')

'eSrcteM seaseg'

>>> crypto('Secret Messages')

'eSrcteM seasegs'

8. Write a function emphasize() that takes as an input a string s and returns (not prints) a string with spaces inserted between adjacent letters.

>>> emphasize('Very important')

'V e r y i m p o r t a n t'

9. Develop function fingerprint() that takes text (as a string) as input and creates and returns the text “fingerprint” obtained as follows: replace each word in the text by its length (i.e., the number of letters) and concatenate these numbers.

>>> fingerprint('This is a secret message')

'42167'

>>> fingerprint('This message has a different fingerprint')

'4731911'

>>> fingerprint('Very Short')

'45'

10. Recall that Roman numerals use symbols M, D, C, X, V, and I whose decimal values are M = 1000, D = 500, C = 100, X = 10, V = 5, I = 1. For example, the Roman numeral MDCXVII corresponds to 1000+500+100+10+5+1+1=1617. There are more complicated rules, e.g. IV usually is 4, but we'll use a simple version of Roman numerals where we just accumulate the values of all symbols. E.g. MIIIMMDCM we'll evaluate as 4\*1000 + 3\*1 + 1\*500 + 1\*100 = 4603. Write a function roman() that takes as an input a string s and evaluates the value of s as a Roman numeral according to these simplified rules.

>>> roman('MDCXVII')

1617

>>> roman('MIIIMMDCM')

4603

11. The following function takes a sentence (i.e., a string) and returns it with every word in the sentence capitalized.

def cap(s):

lst = s.split()

res = ''

for word in lst:

res = res + ' '+ word.capitalize()

return res

We want to modify this function so it takes a sentence and creates a title (for a book, chapter, etc.) Here are the rules we'll follow to capitalize titles:

* The first word is always capitalized.
* Words of length 1 or 2 are not capitalized, and the words 'the' and 'and' are not capitalized (unless any of them are the first word, of course).

>>> cap(‘the life and opinions of tristram shandy, gentleman’)

‘The Life and Opinions of Tristram Shandy, Gentleman’

The first 'the' is upper-case, since it's the first word,  but 'and' is lower-case, as is 'of' since it only has two letters.

12. You can turn a word into pig-Latin by following the following two rules (simplified):

* If the word starts with a consonant, move that letter to the end and append ‘ay’. E.g. ‘happy’ becomes ‘appyhay’ and ‘pencil’ becomes ‘encilpay’.
* If the word starts with a vowel, simply append ‘way’ to the end of the word. E.g. ‘enter’ becomes ‘enterway’ or ‘other’ becomes ‘otherway’. For our purposes, there are 5 vowels: a, e, i, o, u (so we count y as a consonant).

Write a function pig() that takes a word (i.e., a string) as input and returns its pig-Latin form. Your function should still work if the input word is upper case. Your output should always be lower case however.

>>> pig('happy')

'appyhay'

>>> pig('enter')

'enterway'

13. Write a function called honorRoll(). Its only input parameter is a list that contains lists consisting of a name followed by a series of grades (integers). All grades will be in the range 0-100. All students will have at least one grade, but not all students will have the same number of grades. Your function should return a list of students whose average grade is **above** 92.0.

>>> honorRoll([['Alice', 95, 92, 98], ['Carlos', 85, 87, 95, 91], ['Betty', 81, 74, 89]])

['Alice']

>>> honorRoll([['Alice', 92, 92, 92], ['Bob', 81, 46]])

[]

14. In written text small numbers are often written out, e.g. you'd write 'I have two brothers and one sister', rather than 'I have 2 brothers and 1 sister'. In this problem you will implement a function convertSmall() that takes as input a text (i.e., string) s and returns the text s with small numbers (integers between 0 and 6) converted to their names.

>>> convertSmall('I have 2 brothers and 1 sister')

'I have two brothers and one sister'

>>> convertSmall('I have 8 brothers and 5 sisters')

'I have 8 brothers and five sisters'

15. *Encryption* is the process of hiding the meaning of a text by substituting letters in the message with other letters, according to some system. If the process is successful, no one but the intended recipient can understand the encrypted message. *Cryptanalysis* refers to attempts to undo the encryption, even if some details of the encryption are unknown (for example, if an encrypted message has been intercepted). The first step of cryptanalysis is often to build up a table of letter frequencies in the encrypted text. Assume that the string letters is already defined as 'abcdefghijklmnopqrstuvwxyz'. Write a function called frequencies() that takes a string as its only parameter, and returns a list of integers, showing the number of times each character appears in the text. Your function may ignore any characters that are not in letters.

>>> frequencies('The quick red fox got bored and went home.')

[1, 1, 1, 3, 5, 1, 1, 2, 1, 0, 1, 0, 1, 2, 4, 0, 1, 2, 0, 2, 1, 0, 1, 1, 0, 0]

>>> frequencies('apple')

[1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

>>>

16. (while loop) Write a function firstNeg() that takes a (possibly empty) list of numbers as input, finds the first occurrence of a negative number, and returns its index of that number. If there is no negative number or the list is empty, the program should return -1.

>>> firstNeg([2, 3, -1, 4, -2])

2

>>> firstNeg([2, 3, 1, 4, 2])

-1

17. A single playing card can be represented by a string where the first character is a digit 2-9 or a letter ‘A’, ‘T’, ‘J’, ‘Q’, or ‘K’, for Ace, Ten, Jack, Queen, King respectively, and the second character is one of ‘S’, ‘H’, ‘D’, or ‘C’ for Spades, Hearts, Diamonds, or Clubs. A hand of cards can be represented by a list of strings. Write a function called threeOfAKind() that takes a list of strings representing a hand of cards, and returns True if there are exactly three cards with the same value (regardless of suit), False otherwise.

>>> threeOfAKind(['3D', 'KD', 'JD', '3H', '3S'])

True

>>> threeOfAKind(['3D', 'KD', 'JD', '3H', '4S', '5S'])

False

>>> threeOfAKind(['3D', '3H', '3S', '3C', 'AC'])

False

18. Using the same representation of playing cards as the last problem, write a function called flush() that takes a list of strings representing a hand of cards, and returns True if all cards are the same suit, and returns False otherwise.

>>> flush(['AS', '2S', '9S', '6S', 'JS'])

True

>>> flush(['AS', '2S', '9S', '6S', 'JD'])

False

19. Write function season() that takes as input the number of a month (1-12) and prints the season corresponding to the month. You may assume that numbers 1 to 3 correspond to Winter, 4 to 6 correspond to Spring, 7 to 9 correspond to Summer, and 10 to 12 correspond to Fall.

>>> season(3)

Winter

>>> season(11)

Fall

20. The Sieve of Erastophenes is an algorithm -- known to ancient Greeks -- that finds all prime numbers up to a given number n. It does this by first creating a list L from 2 to n and an (initially empty) list primeL. The algorithm then takes the first number in list L (2) and appends it to list primeL, and then removes 2 and all its multiples (4,6,8,10,12, ...) from L. The algorithm then takes the new first number in L (3) and appends it to list primeL, and then removes from L 3 and all its remaining multiples (9,15,21,...). So, in every iteration, the first number of list L is appended to list primeL and then it and its multiples are removed from list L. The iterations stop when list L becomes empty. Write a function sieve() that takes as input a positive integer n, implements the above algorithm, and returns a list of all prime numbers up to n.  
  
>>> sieve(56)  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53]  
>>> sieve(368)  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367]  
>>> sieve(32)  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

21. Write a function digitsum() that takes a positive integer n as input and returns the sum of the digits of n.  
  
>>> digitsum(34)  
7  
>>> digitsum(934)  
16  
>>> digitsum(9314)  
17

22. Write a function innerProduct() that that takes as input two lists (of same length) of integers and then computes and returns the *inner product* of the two lists of integers. The inner product of two lists x = [x1, x2, ..., xn] and y = [y1, y2, ..., yn] is the value x1 y1 +  x2 y2  +  ...  +  xn yn .

>>> innerProduct([2,3,4], [1,0,2])  
10

23. Write a function pairs() that that takes as input two lists of same length x = [x0, x1, x2, ..., xn] and y = [y0, y1, y2, ..., yn] and returns the list [(x1, y1), (x1, y1), (x2, y2), ... , (xn, yn )].  
  
>>> pairs([2,3,4], [1,0,2])  
[(2, 1), (3, 0), (4, 2)]

24. The mathematical constant Pi is an irrational number with value approximately 3.1415928... The precise value of this constant can be obtained from the following inﬁnite sum:

Pi2 = 8 + 8/32 + 8/52 + 8/72 + 8/92 + ...

(Pi is of course just the square root of this value.)   
  
Although we cannot compute the entire inﬁnite series, we get a good approximation of the value of Pi2 by computing the beginning of such a sum. Write a function approxPIsquared() that takes as input float error and approximates constant Pi2 to within error by computing the above sum, term by term, until the difference between the new and the previous sum is less than error. The function should **return** the new sum.   
  
>>> approxPIsquared(0.0001)  
9.855519952254232  
>>> approxPIsquared(0.00000001)  
9.869462988376474