Logic-1 > caught_speeding http://codingbat.com/prob/p137202

You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.

caught_speeding(60, False) → 0
caught_speeding(65, False) → 1
caught_speeding(65, True) → 0

Warmup-2 > array_front9 http://codingbat.com/prob/p110166

Given an array of ints, return True if one of the first 4 elements in the array is a 9. The array length may be less than 4.

array_front9([1, 2, 9, 3, 4]) → True
array_front9([1, 2, 3, 4, 9]) → False
array_front9([1, 2, 3, 4, 5]) → False

- ## Warmup-2 > array123 http://codingbat.com/prob/p193604

Given an array of ints, return True if the sequence of numbers 1, 2, 3 appears in the array somewhere.

array123([1, 1, 2, 3, 1]) → True
array123([1, 1, 2, 4, 1]) → False
array123([1, 1, 2, 1, 2, 3]) → True

List-2 > count_evens http://codingbat.com/prob/p189616

Return the number of even ints in the given array. Note: the % "mod" operator computes the remainder, e.g. 5 % 2 is 1.

count_evens([2, 1, 2, 3, 4]) → 3
count_evens([2, 2, 0]) → 3
count_evens([1, 3, 5]) → 0

List-2 > big_diff http://codingbat.com/prob/p184853

List-2 > centered_average http://codingbat.com/prob/p126968

Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division to produce the final average. You may assume that the array is length 3 or more.

centered_average([1, 2, 3, 4, 100]) → 3
centered_average([1, 1, 5, 5, 10, 8, 7]) → 5
centered_average([-10, -4, -2, -4, -2, 0]) → -3

List-2 > sum13 http://codingbat.com/prob/p167025

Return the sum of the numbers in the array, returning 0 for an empty array. Except the number 13 is very unlucky, so it does not count and numbers that come immediately after a 13 also do not count.

sum13([1, 2, 2, 1]) → 6
sum13([1, 1]) → 2
sum13([1, 2, 2, 1, 13]) → 6

**Array-3 > maxSpan http://codingbat.com/prob/p189576

Consider the leftmost and righmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority.)

maxSpan([1, 2, 1, 1, 3]) → 4
maxSpan([1, 4, 2, 1, 4, 1, 4]) → 6
maxSpan([1, 4, 2, 1, 4, 4, 4]) → 6

**Array-3 > canBalance http://codingbat.com/prob/p158767

Given a non-empty array, return true if there is a place to split the array so that the sum of the numbers on one side is equal to the sum of the numbers on the other side.

canBalance([1, 1, 1, 2, 1]) → true
canBalance([2, 1, 1, 2, 1]) → false
canBalance([10, 10]) → true

***Array-3 > linearIn http://codingbat.com/prob/p134022

Given two arrays of ints sorted in increasing order, **outer** and **inner**, return true if all of the numbers in inner appear in outer. The best solution makes only a single "linear" pass of both arrays, taking advantage of the fact that both arrays are already in sorted order.

linearIn([1, 2, 4, 6], [2, 4]) → true
linearIn([1, 2, 4, 6], [2, 3, 4]) → false
linearIn([1, 2, 4, 4, 6], [2, 4]) → true

***Array-3 > squareUp
http://codingbat.com/prob/p155405

Given n>=0, create an array length n*n with the following pattern, shown here for n=3 : {0, 0, 1, 0, 2, 1, 3, 2, 1} (spaces added to show the 3 groups).

squareUp(3) → [0, 0, 1, 0, 2, 1, 3, 2, 1]
squareUp(2) → [0, 1, 2, 1]
squareUp(4) → [0, 0, 0, 1, 0, 0, 2, 1, 0, 3, 2, 1, 4, 3, 2, 1]

Array-3 > seriesUp http://codingbat.com/prob/p104090

Given n>=0, create an array with the pattern {1, 1, 2, 1, 2, 3, ... 1, 2, 3 .. n} (spaces added to show the grouping). Note that the length of the array will be 1 + 2 + 3 ... + n, which is known to sum to exactly n*(n + 1)/2.

seriesUp(3) → [1, 1, 2, 1, 2, 3]
seriesUp(4) → [1, 1, 2, 1, 2, 3, 1, 2, 3, 4]
seriesUp(2) → [1, 1, 2]

Array-3 > maxMirror http://codingbat.com/prob/p196409

We'll say that a "mirror" section in an array is a group of contiguous elements such that somewhere in the array, the same group appears in reverse order. For example, the largest mirror section in {1, 2, 3, 8, 9, 3, 2, 1} is length 3 (the {1, 2, 3} part). Return the size of the largest mirror section found in the given array.

maxMirror([1, 2, 3, 8, 9, 3, 2, 1]) → 3
maxMirror([1, 2, 1, 4]) → 3 (incorrect? Either 1 or zero?)
maxMirror([7, 1, 2, 9, 7, 2, 1]) → 2

Array-3 > countClumps http://codingbat.com/prob/p193817

Say that a "clump" in an array is a series of 2 or more adjacent elements of the same value. Return the number of clumps in the given array.

countClumps([1, 2, 2, 3, 4, 4]) → 2
countClumps([1, 1, 2, 1, 1]) → 2
countClumps([1, 1, 1, 1, 1]) → 1