Kata Card Game Text

ArtsCow.com design V6 on account for john@techelevator.com

Exercise text: 7pt, Arial, right justified

ToDo:

- Add difficulty ranking

Changed in V6:

Fixed: Edit #25 add array name

Fixed: Replaced #9 in this document with “No14”

Draft 6/26/19 - schedule for Detroit production and direct ship on 6/26/19

Exercise 1: SwapEnds
Create an integer array method called SwapEnds that takes in an integer array "nums". Given an array of ints, swap the first and last elements in the array. Return the modified array. The array length will be at least 1.
For example:
swapEnds([1, 2, 3, 4]) → [4, 2, 3, 1]
swapEnds([1, 2, 3]) → [3, 2, 1]
swapEnds([8, 6, 7, 9, 5]) → [5, 6, 7, 9, 8]

Exercise 2: Has12
Create a boolean method called Has12 that takes in an integer array "nums". Given an array of ints, return true if there is a 1 in the array with a 2 somewhere later in the array.
For example:
has12([1, 3, 2]) → true
has12([3, 1, 2]) → true
has12([3, 1, 4, 5, 2]) → true

Exercise 3: GreenTicket
Create an integer method called GreenTicket that takes in three integers, "a". "b", and "c"
You have a green lottery ticket, with ints a, b, and c on it. If the numbers are all different from each other, the result is 0. If all of the numbers are the same, the result is 20. If two of the numbers are the same, the result is 10.
greenTicket(1, 2, 3) → 0

greenTicket(2, 2, 2) → 20
greenTicket(1, 1, 2) → 10

Exercise 4: Start1
Create an integer method called Start1 that takes in two integer arrays "a" and "b". Start with 2 int arrays, a and b, of any length. Return how many of the arrays have 1 as their first element.
start1([1, 2, 3], [1, 3]) → 2
start1([7, 2, 3], [1]) → 1
start1([1, 2], []) → 1

Exercise 5: FizzArray3
Create an integer array method called FizzArray3 that takes in two integers "start" and "end" Given start and end numbers, return a new array containing the sequence of integers from start up to but not including end, so start=5 and end=10 yields {5, 6, 7, 8, 9}. The end number will be greater or equal to the start number. Note that a length-0 array is valid.
fizzArray3(5, 10) → [5, 6, 7, 8, 9]
fizzArray3(11, 18) → [11, 12, 13, 14, 15, 16, 17]
fizzArray3(1, 3) → [1, 2]

Exercise 6: Only14
Create a boolean method called Only14 that takes in an integer array “nums”. Given an array of ints, return true if every element is a 1 or a 4.
only14([1, 4, 1, 4]) → true
only14([1, 4, 2, 4]) → false
only14([1, 1]) → true

Exercise 7: NoTriples
Create a boolean method called NoTuples that takes in an integer array “nums”. Given an array of ints, we'll say that a triple is a value appearing 3 times in a row in the array. Return true if the array does not contain any triples.
noTriples([1, 1, 2, 2, 1]) → true
noTriples([1, 1, 2, 2, 2, 1]) → false
noTriples([1, 1, 1, 2, 2, 2, 1]) → false

Exercise 8: No23
Create a method of type boolean called No23 that takes in an integer array “nums”. Given an int array length 2, return true if it does not contain a 2 or 3.
no23([4, 5]) → true
no23([4, 2]) → false
no23([3, 5]) → false

Exercise 9: No14

Create a boolean method called No14 that takes in an integer array “nums”. Given an array of ints, return true if it contains no 1's and it contains no 4's.
no14([7, 2, 3]) → true
no14([1, 2, 3, 4]) → false
no14([2, 3, 4]) → false

Exercise 10: More14
Create a boolean method called More14 that takes in an array of type integer “nums”. Given an array of ints, return true if the number of 1's is greater than the number of 4's
more14([1, 4, 1]) → true
more14([1, 4, 1, 4]) → false
more14([1, 1]) → true

Exercise 11: MakeMiddle
Create an integer array method called MakeMiddle that takes in an integer array “nums”. Given an array of ints of even length, return a new array length 2 containing the middle two elements from the original array. The original array will be length 2 or more.
makeMiddle([1, 2, 3, 4]) → [2, 3]
makeMiddle([7, 1, 2, 3, 4, 9]) → [2, 3]
makeMiddle([1, 2]) → [1, 2]

Exercise 12: MakeLast
Create an integer array method called MakeLast that takes in an integer array “nums”. Given an int array, return a new array with double the length where its last element is the same as the original array, and all the other elements are 0. The original array will be length 1 or more. Note: by default, a new int array contains all 0's.
makeLast([4, 5, 6]) → [0, 0, 0, 0, 0, 6]
makeLast([1, 2]) → [0, 0, 0, 2]
makeLast([3]) → [0, 3]

Exercise 13: MakeEnds
Create an integer array method called MakeEnds that takes in an integer array “nums”. Given an array of ints, return a new array length 2 containing the first and last elements from the original array. The original array will be length 1 or more.
makeEnds([1, 2, 3]) → [1, 3]
makeEnds([1, 2, 3, 4]) → [1, 4]
makeEnds([7, 4, 6, 2]) → [7, 2]

Exercise 14: LessBy10
Create a boolean method called LessBy10 that take in three integers, “a, “b”, and “c”. Given three ints, a b c, return true if one of them is 10 or more different than one of the others.
lessBy10(1, 7, 11) → true
lessBy10(1, 7, 10) → false

lessBy10(11, 1, 7) → true

Exercise 15: Has23
Create a boolean method called Has23 that takes in an integer array “nums”. Given an int array length 2, return true if it contains a 2 or a 3.
has23([2, 5]) → true
has23([4, 3]) → true
has23([4, 5]) → false

Exercise 16: FizzArray
Create an integer array method called FizzArray that takes in an integer ”n”. Given a number n, create and return a new int array of length n, containing the numbers 0, 1, 2, ... n-1. The given n may be 0, in which case just return a length 0 array. You do not need a separate if-statement for the length-0 case; the for-loop should naturally execute 0 times in that case, so it just works. The syntax to make a new int array is: new int[desired_length]
fizzArray(4) → [0, 1, 2, 3]
fizzArray(1) → [0]
fizzArray(10) → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Exercise 17: Fix23
Create an integer array called Fix23 that takes in an integer array “nums”. Given an int array length 3, if there is a 2 in the array immediately followed by a 3, set the 3 element to 0. Return the changed array.
fix23([1, 2, 3]) → [1, 2, 0]
fix23([2, 3, 5]) → [2, 0, 5]
fix23([1, 2, 1]) → [1, 2, 1]

Exercise 18: EvenlySpaced
Create a boolean method called EvenlySpaced that takes in three integers, “a”, “b”, and “c”; Given three ints, a b c, one of them is small, one is medium and one is large. Return true if the three values are evenly spaced, so the difference between small and medium is the same as the difference between medium and large.
evenlySpaced(2, 4, 6) → true
evenlySpaced(4, 6, 2) → true
evenlySpaced(4, 6, 3) → false

Exercise 19: Double23
Create a boolean method called Double23 that takes in an integer array “nums”. Given an int array, return true if the array contains 2 twice, or 3 twice. The array will be length 0, 1, or 2.
double23([2, 2]) → true
double23([3, 3]) → true
double23([2, 3]) → false

Exercise 20: BiggerTwo
Create an integer array method called BiggerTwo that takes in two integer arrays “a” and “b”. Start with 2 int arrays, a and b, each length 2. Consider the sum of the values in each array. Return the array which has the largest sum. In event of a tie, return a.
biggerTwo([1, 2], [3, 4]) → [3, 4]
biggerTwo([3, 4], [1, 2]) → [3, 4]
biggerTwo([1, 1], [1, 2]) → [1, 2]

Exercise 21: Blackjack
Create an integer method called Blackjack that takes in two integers, “a” and “b”. Given 2 int values greater than 0, return whichever value is nearest to 21 without going over. Return 0 if they both go over.
blackjack(19, 21) → 21
blackjack(21, 19) → 21
blackjack(19, 22) → 19

Exercise 22: ArrayFront9
Create a boolean method called ArrayFront9 that takes in an integer array “nums”. 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.
arrayFront9([1, 2, 9, 3, 4]) → true
arrayFront9([1, 2, 3, 4, 9]) → false
arrayFront9([1, 2, 3, 4, 5]) → false

Exercise 23: ArrayCount9
Create an integer method called ArrayCount9 that takes in an integer array “nums”. Given an array of ints, return the number of 9's in the array.
arrayCount9([1, 2, 9]) → 1
arrayCount9([1, 9, 9]) → 2
arrayCount9([1, 9, 9, 3, 9]) → 3

Exercise 24: Array123
Create a boolean method called Array123 that takes in an integer array “nums”. Given an array of ints, return true if .. 1, 2, 3, .. appears in the array somewhere in order.
array123([1, 1, 2, 3, 1]) → true
array123([1, 1, 2, 4, 3]) → false
array123([1, 1, 2, 1, 2, 3]) → true

Exercise 25: IsStrictlyIncreasing
Create a boolean method called IsStrictlyIncreasing that takes in an integer array “nums”. Given an array of integers, return true if the values are strictly increasing. Return false otherwise
isStrictlyIncreasing([5,7,8,10]) → true
isStrictlyIncreasing([5,7,7,10]) → false

isStrictlyIncreasing([-5,-3,0,17]) → true

Exercise 26: SumOdds
Create an integer method that takes in no arguments or parameters. Return the sum of the odd integers between 1 and 100 inclusive.
sumOdds() → 2500

Exercise 27: SumOddsBetweenValues
Create an integer method that takes in two integer arguments “start” and “end”. Return the sum of the odd integers between “start” and “end” inclusive. “End” will not be less than “start”.
sumOddsBetweenValues(0, 5) → 9
sumOddsBetweenValues(28,30) → 29
sumOddsBetweenValues(18, 18) → 0

Exercise 28: FizzBuzz
Create a string array method called FizzBuzz that takes in no arguments or parameters. Create a method that returns a string array with a string for each integer from 1 to 100 inclusive. If the value is an even multiple of 3, put the string “Fizz” in the aray. Even multiple of 5, put in “Buzz”. Even multiple of both 3 and 5, put in “FizzBuzz”. For all other values, put the number in the resulting array.
fizzBuss() → ([1,2,Fizz,4,Buzz, Fizz,7,8,Fizz,Buzz,11,...])

Exercise 29: Fibonacci (new for V3)
Create an integer array method called Fibonacci that takes in no arguments or parameters. In a Fibonacci sequence, every number after the first two is the sum of the two preceding ones. Create a method that returns a Fibonacci sequence that begins 0,1,1,2,3,.... for the values less than 2000.
fibonacci () → ([1,1,2,3,5,8,13,....,987,1597])

Exercise 30: ReverseString (new for V3)
Create a string method called ReverseString that takes in a string “str”. Create a method that returns a string in the reverse order. The string may be empty, but not null.
reverseString(“Hello!”) → (“!olleH”)
reverseString(“Kata”) → (“ataK”)
reverseString(“”) → (“”)

Exercise 31: PrimeFactors (new for V3)
Create an integer array method called PrimeFactors that take in an integer “n” and returns an integer array of the prime factors of the number. The input will be greater than 1.
primeFactors(6) → ([2,3])
primeFactors(28) → ([2,2,7])
primeFactors(667) → ([23, 29])

Exercise 32: Factorial (new for V3)
Create an integer method called Factorial that takes in an integer “n” and returns the factorial of the number. If the integer is represented with the letter n, a factorial (n!) is the product of all positive integers less than or equal to n.
factorial(3) → (6)
factorial(4) → (24)
factorial(10) → (3628800)

Exercise 33: ComboString (new for V4)
Create a string method called ComboString that takes in two strings, “a” and “b”.
Given 2 strings, a and b, return a string of the form short+long+short, with the shorter string on the outside and the longer string on the inside. The strings will not be the same length, but they may be empty (length 0).
combostring("Hello", "hi") → "hiHellohi"
combostring("hi", "Hello") → "hiHellohi"
combostring("aaa", "b") → "baaab"

Exercise 34: HelloName (new for V4)
Create a string method called HelloName that takes in a string “name”.
Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".
helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Exercise 35: HasBad (new for V4)
Create a boolean method called HasBad that takes in a string “str”. Given a string, return true if "bad" appears starting at index 0 or 1 in the string, such as with "badxxx" or "xbadxx" but not "xxbadxx". The string may be any length, including 0.
hasBad("badxx") → true
hasBad("xbadxx") → true
hasBad("xxbadxx") → false

Exercise 36: FrontTimes (new for V4)
Create a string method called FrontTimes that takes in a string “str” and an integer “n”. Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;
frontTimes("Chocolate", 2) → "ChoCho"
frontTimes("Chocolate", 3) → "ChoChoCho"
frontTimes("Abc", 3) → "AbcAbcAbc"

Exercise 37: FirstTwo (new for V4)

Create a string method called FirstTwo that takes in a string “str”. Given a string, return the string made of its first two chars, so the string "Hello" yields "He". If the string is shorter than length 2, return whatever there is.
firstTwo("Hello") → "He"
firstTwo("abcdefg") → "ab"
firstTwo("ab") → "ab"

Exercise 38: FirstHalf (new for V4)
Create a string method called FirstHalf that takes in a string “str”. Given a string of even length, return the first half. So the string "WooHoo" yields "Woo".
firstHalf("WooHoo") → "Woo"
firstHalf("HelloThere") → "Hello"
firstHalf("abcdef") → "abc

Exercise 39: ExtraEnd (new for V4)
Create a string method called ExtraEnd that takes in a string “str”. Given a string, return a new string made of 3 copies of the last 2 chars of the original string. The string length will be at least 2.
extraEnd("Hello") → "lololo"
extraEnd("ab") → "ababab"
extraEnd("Hi") → "HiHiHi"

Exercise 40: EndsLy (new for V4)
Create a boolean method called EndsLy that takes in a string “str”. Given a string, return true if it ends in "ly".
endsLy("oddly") → true
endsLy("y") → false
endsLy("oddy") → false

Exercise 41: CountXX (new for V4)
Create an integer method called CountXX that takes in a string “str”. Count the number of "xx" in the given string. We'll say that overlapping is allowed, so "xxx" contains 2 "xx".
countXX("abcxx") → 1 --->
countXX("xxx") → 2
countXX("xxxx") → 3

Exercise 42: DoubleX (new for V4)
Create a boolean method called DoubleX that takes in a string “str”. Given a string, return true if the first instance of "x" in the string is immediately followed by another "x".
doubleX("axxbb") → true
doubleX("axaxax") → false
doubleX("xxxxx") → true

Exercise 43: ReverseList
Create a list of type string method called ReverseList that takes in a list of type string “strings”. Given a list of type string, return a new list in reverse order of the original. (Hint: Think Stack)
reverseList(["purple", "green", "blue", "yellow", "green"]) → ["green", "yellow", "blue", "green", "purple"]
reverseList(["jingle", "bells", "jingle", "bells", "jingle", "all", "the", "way"})
→ ["way", "the", "all", "jingle", "bells", "jingle", "bells", "jingle"]

Exercise 44: OddOnly (new for V4)
Create a list of type integer method called OddOnly that takes in an integer array “nums”. Given an array of integers, return a list of integers containing just the odd values.
oddOnly({112, 201, 774, 92, 9, 83, 41872}) → [201, 9, 83]
oddOnly({1143, 555, 7, 1772, 9953, 643}) → [1143, 555, 7, 9953, 643]
oddOnly({734, 233, 782, 811, 3, 9999}) → [233, 811, 3, 9999]

Exercise 45: No4LetterWords (new for V4)
Create a list of type string method called No4LetterWords that takes in an array of type string “strings”. Given an array of strings, return a list containing the same strings in the same order except for any words that contain exactly 4 characters.
no4LetterWords({"Train", "Boat", "Car"}) → ["Train", "Car"]
no4LetterWords({"Red", "White", "Blue"}) → ["Red", "White"]

Exercise 46: List2Array (new for V4)
Create an array of type string method called List2Array that takes in a list of type string “strings”. Given a list of strings, return an array containing the same Strings in the same order. Avoid using a ToArray method.
list2Array(["aa", "ab", "ac"]) → {"aa", "ab", "ac"}
list2Array(["as", "df", "jk"]) → {"as", "df", "jk"}
list2Array(["aaa", "bbb", "ccc", "ddd"]) → {"aaa", "bbb", "ccc", "ddd"}

Exercise 47: FoundIntTwice (new for V4)
Create a boolean method called FoundIntTwice that takes in a list of type integer “nums” and an integer “value”. Given a list of Integers, and an int value, return true if the int value appears two or more times in the list.
foundIntTwice([5, 7, 9, 5, 11], 5)→ true
foundIntTwice([6, 8, 10, 11, 13], 8→ false
foundIntTwice([9, 23, 44, 2, 88, 44], 44)→ true

Exercise 48: FindLargest (new for V4)

Create an integer method called FindLargest that takes in an integer list “nums”. Given a list of integers, return the largest value. If there is a Max method, can you do this without using the method?
findLargest([11, 200, 43, 84, 9917, 4321, 1, 33333, 8997]) → 33333
findLargest([987, 1234, 9381, 731, 43718, 8932]) → 43718
findLargest([34070, 1380, 81238, 7782, 234, 64362, 627]) → 64362

Exercise 49: DistinctValues (new for V4)
Create a list of type string method called DistinctValues that takes in a list of type string “strings”. Given a list of strings, return a list that contains the distinct values. (Hint: Think HashSet)
distinctValues(["red", "yellow", "green", "yellow", "blue", "green", "purple"]) → ["red", "yellow", "green", "blue", "purple"]
distinctValues(["jingle", "bells", "jingle", "bells", "jingle", "all", "the", "way"]) → ["jingle", "bells", "all", "the", "way"]

Exercise 50: Array2List (new for V4)
Create a list of type string method called Array2List that takes in a string array “strings”. Given an array of strings, return a List containing the same Strings in the same order
array2List({"Apple", "Orange", "Banana"}) → ["Apple", "Orange", "Banana"]
array2List({"Red", "Orange", "Yellow"}) → ["Red", "Orange", "Yellow"]
array2List({"Left", "Right", "Forward", "Back"}) → ["Left", "Right", "Forward", "Back"]

Exercise 51: ArrayInt2ListDouble (new for V4)
Create a list of type double method called ArrayInt2ListDouble that takes in an integer array “ints”. Given an array of ints, divide each int by 2, and return a list of doubles.
arrayInt2ListDouble({5, 8, 11, 200, 97}) → [2.5, 4.0, 5.5, 100, 48.5]
arrayInt2ListDouble({745, 23, 44, 9017, 6}) → [372.5, 11.5, 22, 4508.5, 3]
arrayInt2ListDouble({84, 99, 3285, 13, 877}) → [42, 49.5, 1642.5, 6.5, 438.5]

Exercise 52: MakeOutWord (new for V4)
Create a string method called MakeOutWord that takes in two strings, “outword” and “word”. Given an "outword" string length 4, such as "<<>>", and a word, return a new string where the word is in the middle of the outword string, e.g. "<<word>>".
makeOutWord("<<>>", "Yay") → "<<Yay>>"
makeOutWord("<<>>", "WooHoo") → "<<WooHoo>>"
makeOutWord("[[]]", "word") → "[[word]]"

Kata- A Game for New Coders
A "party" game for junior level developers named "Kata" . Teams or individual. Each player draws a card with the description of a problem to solve. Player "whiteboards" solution. Other players determine if the solution is correct. Students should be able to play this any time after week 2 of the Tech Elevator program.

Suggestions to: John Fulton (john@techelevator.com)
Special thanks to:
Kalyn Breneman: Design, branding and encouragement
Tech Elevator Instructor Staff: Development of the exercises used on the cards
ArtsCow.com production staff in Hong Kong: Extra assistance with initial printing runs
Will Stevens & Alex Khrizma: Production sourcing, testing and gamification ideas