14. Tests 1-3 Review

Github Repository | Vimeo Video Recording

Test 1

Question 0

Implement the functions defined below

COUNT - the number of items in a list

For example: `count([6,2,3,4,9,6,1,0,5]);` Returns: `9`

```
const count = function (arr) {
  /* IMPLEMENT ME */
};
```

SUM - the sum of the numbers in a list - safe to assume that all items are numbers already

For example: `sum([6,2,3,4,9,6,1,0,5])`

Returns: `36`

```
const sum = function (arr) {
  /* IMPLEMENT ME */
};
```

To be used by mean. Do not alter.

```
const round = function (number) {
  return Math.round(number * 100) / 100;
};
```

MEAN - the average value of numbers in a list - use the provided 'round' function when returning your value - if empty array, return null to indicate that mean cannot be calculated

For example: `mean([6,2,3,4,9,6,1,0,5])`

Returns: `4`

```
const mean = function (arr) {
  /* IMPLEMENT ME */
};
```

Question 1

MIN - the lowest value in a list For example: `min([6,2,3,4,9,6,1,0,5])` Returns: `0`

```
const min = function (arr) {
  /* IMPLEMENT ME */
};
```

MAX - the highest value in a list For example: `max([6,2,3,4,9,6,1,0,5])` Returns: `9`

```
const max = function (arr) {
  /* IMPLEMENT ME */
};
```

RANGE - the difference between the highest and lowest values in a list For example: `range([6,2,3,4,9,6,1,0,5])` Returns: `9`

```
const range = function (arr) {
  /* IMPLEMENT ME */
};
```

Question 2

// Meant to be used by median. Do not alter.

```
const round = function (number) {
  return Math.round(number * 100) / 100;
};
```

MEDIAN - the middle number of a list (when sorted) - if the list length is even, then the median is the average of the two middle values - use the provided 'round' function before returning your value For example: `median([6,2,3,4,9,6,1,0,5]);` Returns: `4`

```
const median = function (arr) {
  /* IMPLEMENT ME */
};
```

Question 3

Implement the 'mode' function defined below

MODE - the most frequently occurring number - for this test, the provided lists will only have a single value for the mode For example: `mode([6,2,3,4,9,6,1,0,5]);` Returns: `6`

```
const mode = function (arr) {
  /* IMPLEMENT ME */
};
```

Question 4

Implement the 'stdev' function defined below STDEV - the square root of the average of the squared deviations of the values from their average value - The formula is: stdev = sqrt(sum((x - populationMean)^2)/numberOfValues) - you are allowed to look at Wikipedia's example calculation to help you understand the formula - Keep in mind, we are using 'Population Standard Deviation' as opposed to 'Sample Standard Deviation' for this testhttps://en.wikipedia.org/wiki/Standard_deviation#Population_standard_deviation_of_grades_of_eight_students - use the provided 'round' function before returning your final value - you can take a square root using `Math.sqrt(number)` For example: `stdev([6,2,3,4,9,6,1,0,5]);` Returns: `2.67`

// This function is to be used by stdev. Do not alter. const round = function(number) { return Math.round(number * 100) / 100; };

```
const stdev = function (arr) {
  /* IMPLEMENT ME */
};
```

Test 2

Question 0

Write a converter that will change Celsius to Fahrenheit and back again.

Your function should take in a number, and a boolean. The number will be the temperature in degrees, and the boolean will be whether to convert from C to F (true) or F to C (false). Your answer should be rounded to one decimal place, and returned as a Number, not a string.

If the first argument is not a number, return NaN for the result.

Examples:

- tempConverter(32, true) returns 89.6 as a result
- tempConverter(32, false) returns 0.0 as a result
- tempConverter(98.6, false) returns 37 as a result
- tempConverter("12",) returns NaN as a result

```
const tempConverter = function (value, cToF) {
  //implement me
};
```

Question 1

Build a function called keyMatcher() which, when passed two objects and a string, will use the string to look up the key-value pair in each object and compare the values. If the two values are explicitly equal to each other, return true, otherwise return false if either the values or not the same, or both objects do not have that key.

Examples:

- `keyMatcher({a: 1, b: 2, c: 3}, {here: 3, is: 2, a: 1, silly: 0, example: -1}, 'a')` returns `true` (since the value and type are the exact same)
- `keyMatcher({apple: "red", banana: "yellow", cherry: "red"}, {apple: "green", banana: "brown", cherry: "black"}, "apple")` returns `false` since the values are completely different ("red" vs "green")
- `keyMatcher({a: 1, b: 2, c: 3}, {a: "1", b: "2", c: "3"}, 'c')` returns `false` since the values are different types (3 vs "3")
- `keyMatcher({a: 1, b: 2, c: 3}, {d: 4, e: 5, f: 6}, 'b')` returns `false` since b is not in the second object

```
const keyMatcher = function (firstObj, secondObj, key) {
  /* IMPLEMENT ME */
};
```

Question 2

Implement a function called countWhich() which will take in a list of items and a callback, and it will return the number of elements which return a truthy value from the callback function.

If the first argument is not an array, our function should return false instead of a number.

Examples:

- `countWhich([1, 2, 3, 4, 5], function(num) { return (num > 4); })` returns `1` (only matches 5)
- `countWhich(["apple", "banana", "cherry"], function(fruit) { return fruit[0] === "a"; })` returns `1` (only matches apple)
- `countWhich([10, 20, 30, 40, 50], function(num) { return num % 7 === 0; })` returns `0` (none of the numbers are divisible by 7)
- `countWhich(["apple", "banana", "cherry"], function(fruit) { return fruit.length > 5; })` returns `2` ("apple" is shorter than 6 characters)
- `countWhich([], function(x) { return x > 10 })` returns `0`
- `countWhich("This should fail", function(word) { return true; })` returns `false` (because the first argument is not an array)

```
const countWhich = function (list, cb) {
  // implement me
};
```

Question 3

Implement the function as defined below.

Many programming languages have a range() functionality which will generate an array of numbers that increment from either 0 or 1 up to the number of digits requested. For example, range(10) might return [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] or it might return [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] depending on the implementation. Some even allow you to control the direction.

Unfortunately, JavaScript doesn't have a built-in range() function, so we want you to build one. Build out the function range() so that it takes three parameters:

1. The number of integers to generate
2. A boolean for whether to skip 0 or not (true: skip zero)

3. A boolean for whether the range should be in reverse/decreasing order or regular/ increasing order (true: reverse/decreasing order)

If a non-number is passed in for the first argument, return an empty array.

Pro Tip: Remember to work incrementally. Start off just implementing the false and false scenario for the second and third parameters. In other words, focus on the zero-based, ascending range first. Work on edge cases at the very end (such as passing in a string instead of a number, as shown in the final example below.)

Examples:

- range(10, false, false) should return [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- range(10, true, false) should return [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- range(10, true, true) should return [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
- range(10, false, true) should return [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
- range(3, true, false) should return [1, 2, 3]
- range(0, false,) should return [0]
- range(10) should do the same thing as range(10, false, false)
- range(10, true) should do the same thing as range(10, true, false)
- range("2", ,) should return []

```
const range = function (count, skipZero, descending) {
  //implement me
};
```

Question 4

Implement the function as defined below.

Given an array of values, the minmax() function will return an array that contains the minimum and maximum values in the array, always with minimum at index 0 and maximum at index 1. For the purposes of this question, you are not allowed to use Math.max() or Math.min().

The array can be a list of lower-cased strings instead of numbers. In this case, min is the string that would be sorted first alphabetically and max is the string that would be sorted last alphabetically ("a" < "b", while "ab" > "aa", and so on).

Mixed-type (strings and numbers) arrays are not of concern to us (meaning, do not worry about this situation).

Examples:

- minmax([1, 2, 3, 4, 5]) returns [1, 5]
- minmax([90, 89, 123, 3]) returns [3, 123]
- minmax(["apple", "banana", "canada"]) returns ["apple", "canada"]
- minmax([]) returns [undefined, undefined]

```
const minmax = function (list) {
  // implement me
};
```

Test 3

Question 0

Convert a given object into an array of arrays.

Given an object, create an array which contains arrays with the key/value pairs.

To keep this simple, the values will only be primitive types (number, string, etc.) and not other objects or array.

Furthermore, assume that the input is always clean/accurate. In other words, don't worry about handling edge cases.

Examples

`objectToArray({ a: 1, b: 2, c: 3 }) => [['a', 1], ['b', 2], ['c', 3]]``objectToArray({name: 'Dave', role: 'Instructor', yearsOfExperience: 10}) => [['name', 'Dave'], ['role', 'Instructor'], ['yearsOfExperience', 10]]`

```
const objectToArray = function (obj) {
  // implement me
};
```

Question 1

Convert an array of arrays into an object.

This is the inverse of the previous question.

Arrays will only have two elements: [key, value]

To keep this simple, the values will only be primitive types (number, string, etc.) and not other objects or array.

Furthermore, assume that the input is always clean/accurate. In other words, don't worry about handling edge cases.

Examples:

- arrayToObject([['a', 1], ['b', 2], ['c', 3]]) => { a: 1, b: 2, c: 3 }
- arrayToObject([['name', 'Dave'], ['role', 'Instructor'], ['yearsOfExperience', 10]]) => {name: 'Dave', role: 'Instructor', yearsOfExperience: 10}

```
const arrayToObject = function (arr) {
  // IMPLEMENT ME
};
```

Question 2

Write a function which will split an array into two arrays (i.e. partition it).

It will take two parameters, the first is an array of Integer values, and the second will be a callback which will return a boolean. If the callback returns true for an element, it should be placed into the left array, otherwise it should be placed into the right array.

Examples:

- partition([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], n => n % 2 === 0) => [[2, 4, 6, 8, 10], [1, 3, 5, 7, 9]]
- partition([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], n => n < 0) => [[-5, -4, -3, -2, -1], [0, 1, 2, 3, 4, 5]]

```
const partition = function (arr, callback) {
  // IMPLEMENT ME
};
```

Question 3

This is a STRETCH QUESTION.

Let's revisit Question 01 which had us convert an array of arrays into an object.

This time, make it support nested arrays.

The nested arrays also only contain 2 element arrays to represent key & value: [key, value]. However, this time we are allowing the value to be another array.

Non-array objects need NOT be supported/handled at all.

Examples:

- deepArrayToObject([['a', 1], ['b', 2], ['c', 'd', 4]]) => { a: 1, b: 2, c: { d: 4 } }
- deepArrayToObject([['a', 1], ['b', 2], ['c', [['d', [['e', 5], ['f', 6]]]]]]) => { a: 1, b: 2, c: { d: { e: 5, f: 6 } } }

```
const deepArrayToObject = function (arr) {
  // IMPLEMENT ME
};
```

Question 4

This is a STRETCH QUESTION.

Given a size in bits convert it to relevant size in B/KB/MB/GB/TB. Round your answers to two decimal places at most. Use base 10 for sizes.

- 1 B
- 1 kB == 1000 B
- 1 MB == 1000 kB
- 1 GB == 1000 MB
- 1 TB == 1000 GB

More info on these in case you are curious:https://en.wikipedia.org/wiki/Byte#Unit_symbol

Examples:

- filesize(1) => "1B"
- filesize(1000) => "1kB"
- filesize(1000000) => "1MB"
- filesize(1500000) => "1.5MB"
- filesize(1250000000) => "1.25GB"
- filesize(9000000000000) => "9TB"

```
const filesize = function (bytes) {
  // IMPLEMENT ME
};
```

View on Compass | Leave Feedback