Basic Algorithm Scripting

An algorithm is a series of step-by-step instructions that describe how to do something.

To write an effective algorithm, it helps to break a problem down into smaller parts and think carefully about how to solve each part with code.

In this course, you'll learn the fundamentals of algorithmic thinking by writing algorithms that do everything from converting temperatures to handling complex 2D arrays.

Convert Celsius to Fahrenheit

The formula to convert from Celsius to Fahrenheit is the temperature in Celsius times `9/5`, plus `32`.

You are given a variable `celsius` representing a temperature in Celsius. Use the variable `fahrenheit` already defined and assign it the Fahrenheit temperature equivalent to the given Celsius temperature. Use the formula mentioned above to help convert the Celsius temperature to Fahrenheit.

Solution:

```
function convertCtoF(celsius) {
  let fahrenheit = celsius * 9/5 +32;
  return fahrenheit;
}

console.log(convertCtoF(30));
```

Reverse a String

Reverse the provided string and return the reversed string.

For example, `"hello"` should become `"olleh"`.

Solution:

```
function reverseString(str) {
  let reversedStr = "";
  for (let i = str.length - 1; i >= 0; i--) {
    reversedStr += str[i];
```

```
  }
  return reversedStr;
}
console.log(reverseString("hello"));
```

Factorialize a Number

Return the factorial of the provided integer.

If the integer is represented with the letter `n`, a factorial is the product of all positive integers less than or equal to `n`.

Factorials are often represented with the shorthand notation `n!`

For example: `5! = 1 * 2 * 3 * 4 * 5 = 120`

Only integers greater than or equal to zero will be supplied to the function.

Solution:

```
function factorialize(num) {
  let product = 1;
  for (let i = 2; i <= num; i++) {
    product *= i;
  }
  return product;
}

factorialize(5);
```

Find the Longest Word in a String

Return the length of the longest word in the provided sentence.

Your response should be a number.

Solution:

```
function findLongestWordLength(str) {
let longestLength = 0;
let currentLength = 0;

for (let i = 0; i < str.length; i++) {
  if (str[i] === " ") {
    if (currentLength > longestLength) {
      longestLength = currentLength;
    }
    currentLength = 0;
  } else {
    currentLength++;
  }
}
if (currentLength > longestLength) {
  longestLength = currentLength;
}

return longestLength;
}
```

Return Largest Numbers in Arrays

Return an array consisting of the largest number from each provided sub-array. For simplicity, the provided array will contain exactly 4 sub-arrays.

Remember, you can iterate through an array with a simple for loop, and access each member with array syntax `arr[i]`.

Solution:

```
function largestOfFour(arr) {
  return arr.map(subArr => {
    return Math.max(...subArr);
  });
```

```
}

largestOfFour([[4, 5, 1, 3], [13, 27, 18, 26], [32, 35, 37, 39], [1000, 1001, 857, 1]]);
```

Confirm the Ending

Check if a string (first argument, `str`) ends with the given target string (second argument, `target`).

This challenge *can* be solved with the `.endsWith()` method, which was introduced in ES2015. But for the purpose of this challenge, we would like you to use one of the JavaScript substring methods instead.

Solution:

```
function confirmEnding(str, target) {
  // "Never give up and good luck will find you."
  // -- Falcor

  return str.slice(str.length - target.length) === target
;
}

confirmEnding("He has to give me a new name", "name");
```

Repeat a String Repeat a String

Repeat a given string `str` (first argument) for `num` times (second argument). Return an empty string if `num` is not a positive number. For the purpose of this challenge, do *not* use the built-in `.repeat()` method.

Solution:

```
function repeatStringNumTimes(str, num) {
  let accumulatedStr = "";

  for (let i = 0; i < num; i++) {
    accumulatedStr += str;
  }

  return accumulatedStr;
}

repeatStringNumTimes("abc", 3);
```

Truncate a String

Truncate a string (first argument) if it is longer than the given maximum string length (second argument). Return the truncated string with a `...` ending.

Solution:

```
function truncateString(str, num) {
  // Clear out that junk in your trunk
  if (str.length > num) {
    return str.slice(0, num) + "...";
  } else {
    return str;
  }
}
```

Finders Keepers

Create a function that looks through an array `arr` and returns the first element in it that passes a 'truth test'. This means that given an element `x`, the 'truth test' is passed if `func(x)` is `true`. If no element passes the test, return `undefined`.

Solution:

```
function findElement(arr, func) {
  let num = 0;
```

```
  for (let i = 0; i < arr.length; i++) {
    num = arr[i];
    if (func(num)) {
      return num;
    }
  }

  return undefined;
}
findElement([1, 2, 3, 4], num => num % 2 === 0);
```

Title Case a Sentence

Return the provided string with the first letter of each word capitalized. Make sure the rest of the word is in lower case.

For the purpose of this exercise, you should also capitalize connecting words like `the` and `of`.

Solution:

```
function titleCase(str) {
  const newTitle = str.split(" ");
  const updatedTitle = [];
  for (let st in newTitle) {
    updatedTitle[st] = newTitle[st][0].toUpperCase() + ne
wTitle[st].slice(1).toLowerCase();
  }
  return updatedTitle.join(" ");
}

console.log(titleCase("I'm a little tea pot"));
```

Slice and Splice

You are given two arrays and an index.

Copy each element of the first array into the second array, in order.

Begin inserting elements at index `n` of the second array.

Return the resulting array. The input arrays should remain the same after the function runs.

Solution:

```
function frankenSplice(arr1, arr2, n) {
  // It's alive. It's alive!
  let localArray = arr2.slice();
  for (let i = 0; i < arr1.length; i++) {
    localArray.splice(n, 0, arr1[i]);
    n++;
    console.log(localArray)
  }
  return localArray;
}


console.log(frankenSplice([1, 2, 3], [4, 5, 6], 1));
```

Falsy Bouncer

Remove all falsy values from an array. Return a new array; do not mutate the original array.

Falsy values in JavaScript are `false`, `null`, `0`, `""`, `undefined`, and `NaN`.

Hint: Try converting each value to a Boolean.

Solution:

```
function bouncer(arr) {
  const filteredArr = [];
```

```
  for (let i = 0; i < arr.length; i++) {
    if (arr[i]) filteredArr.push(arr[i]);
  }
  return filteredArr;
}

console.log(bouncer([7, "ate", "", false, 9]));
```

Where do I Belong

Return the lowest index at which a value (second argument) should be inserted into an array (first argument) once it has been sorted. The returned value should be a number.

For example, `getIndexToIns([1,2,3,4], 1.5)` should return `1` because it is greater than `1` (index 0), but less than `2` (index 1).

Likewise, `getIndexToIns([20,3,5], 19)` should return `2` because once the array has been sorted it will look like `[3,5,20]` and `19` is less than `20` (index 2) and greater than `5` (index 1).

Solution:

```
function getIndexToIns(arr, num) {
  arr.sort((a, b) => a - b);

  for (let i = 0; i < arr.length; i++) {
    if (arr[i] >= num) return i;
  }

  return arr.length;
}

getIndexToIns([40, 60], 50);
```

Mutations

Return `true` if the string in the first element of the array contains all of the letters of the string in the second element of the array.

For example, `["hello", "Hello"]`, should return `true` because all of the letters in the second string are present in the first, ignoring case.

The arguments `["hello", "hey"]` should return `false` because the string `hello` does not contain a `y`.

Lastly, `["Alien", "line"]`, should return `true` because all of the letters in `line` are present in `Alien`.

Solution:

```
function mutation(arr) {
  const test = arr[1].toLowerCase();
  const target = arr[0].toLowerCase();
  for (let i = 0; i < test.length; i++) {
    if (target.indexOf(test[i]) < 0) return false;
  }
  return true;
}
```

Chunky Monkey

Write a function that splits an array (first argument) into groups the length of `size` (second argument) and returns them as a two-dimensional array.

Solution:

```
function chunkArrayInGroups(arr, size) {
  let temp = [];
  const result = [];

  for (let a = 0; a < arr.length; a++) {
    if (a % size !== size - 1) temp.push(arr[a]);
    else {
```

```
      temp.push(arr[a]);
      result.push(temp);
      temp = [];
    }
  }

  if (temp.length !== 0) result.push(temp);
  return result;
}
```