W1D1 – Capitalise the Front

Using for loop

const capitaliseTheFront = (string) => {

// Declare empty array to store answer

answer = [];

// Convert string to lower case and split into individual elements to iterate over

newArray = string.toLowerCase().split(' ');

console.log(newArray);

//loop through to capitalise the 1st letter of each word

for (let i=0; i < newArray.length; i++) {

answer.push(newArray[i].charAt(0).toUpperCase() + newArray[i].substr(1));

console.log(answer);

}

return answer.join(' ');

}

//Using For Each

const capitaliseTheFront = (string) => {

// split string into array of separate words by splitting on the space. Assign array to words variable

answer = [];

words = string.split(' ');

//iterate over words array & for each word, upcase 1st ch & downcase the rest

words.forEach((i) => {

answer.push(i[0].toUpperCase() + i.substr(1).toLowerCase())

})

// join elements in new array with space as seperator

return answer.join(' ');

}

// Using MAP

const capitaliseTheFront = (string) => {

result = [];

string.split(" ").map((word) => {

result.push(word.charAt(0).toUpperCase() + word.slice(1).toLowerCase());

})

return result.join(" ")

}

W1D2 – Common Characters

// Using Set removes all repeating char and divide individual elements of the set

function commonCharacters(string1, string2) {

/\* Create new set for string1. Use global function that takes 2 arg, 1st find spaces

& 2nd tell it what to do - strip spaces. Also convert to lowercase\*/

set = new Set(string1.replaceAll(' ', '').toLowerCase());

// Create empty string to iterate over set

commonChars = '';

// Use ch to rep character in set

for (let i of set) {

/\* Check if character is included in string2, if yes add/append to commonChars

string & apply lower case \*/

if (string2.toLowerCase().includes(i)) {

commonChars += i;

}

}

return commonChars;

}

// Using for of loop

function commonCharacters(string1, string2) {

answer = [];

// Convert to lowercase and replace space with no space

array1 = string1.toLowerCase().replace(/\s/g, '');

array2 = string2.toLowerCase().replace(/\s/g, '');

for (let char of array1) {

// Compare array1 check if same char is in array2 if yes is it included in answer? If not push into answer array

if(array2.includes(char) && !answer.includes(char)) {

answer.push(char)

console.log(answer);

};

}

return answer.join('');

}W2D1 – Multiple Arguments  
  
// Reduce method

const multipleArguments = (...args) => {

// If args = positive number return args reduced to single value

if (args.length > 0) {

// Iterate over array for every pair of 2 elements (prev, curr placeholder for values) do something

return args.reduce((prev, curr) => prev \* curr);

}

// If no arg entered return 0

return 0;

}

// ...arg rest syntax allow anything pass in as argument

// const multipleArguments = (...args) => {

// // check if args array is empty, if so return 0

// if (args.length === 0) {

// return 0;

// } else {

// // iterate over array of args and \* 1 by 1

// let result = 1;

// for (let num of args) {

// result \*= num;

// }

// return result;

// }

// }

W2D2 – Atomic Blonde

// Use map  
const atomicBlonde = (num) => {

// toString function turn num into string. If pass in num it will turn num to diff base.

// split into empty string it will return array of seperate digits

// map over array providing Number as arg to return similar array of numbers

let digits = num.toString().split('').map(Number);

// create variables 1 is neutral base for mutiply

let add = 0;

let mult = 1;

// Iterate over array change add and mult by adding every num to add and \* every num to mult

// by the end of iteration/array end up with sum of all numbers

for (let i of digits) {

add += i;

mult \*= i;

}

// return compare add and mult, if = return true diff return false.

return add === mult;

}

// const atomicBlonde = (num) => {

// // Array.from create new copy of Array instance

// let digits = Array.from(String(num), Number);

// // Create sum variable and add numbers

// let sum = digits.reduce((prev, curr) => prev + curr);

// // Create product variable and \* numbers

// let product = digits.reduce((prev, curr) => prev \* curr);

// return sum === product;

// }

//Use Array.from and reduce

// const atomicBlonde = (num) => {

// // Array.from creates a new shallow-copied Array instance from an iterable passing in element, index

// let digits = Array.from(String(num), Number);

// // callback function on each element of the array, in order, passing in the return value

// // from the calc on the preceding element. The final result running the reducer across all elements of the array=single value.

// let sum = digits.reduce((prev, cur) => prev + cur)

// let product = digits.reduce((prev, cur) => prev \* cur)

// return sum === product

// }

// Using parseInt function

// const atomicBlonde = (num) => {

// // declare varables sum and product to store answer. Product can't be 0 when \*

// let sum = 0;

// let product = 1;

// let digits = String(num).split('')

// // loop through digit array

// for (let digit of digits) {

// // parseInt parses a string argument and returns an integer of the specified

// sum += parseInt(digit)

// product \*=parseInt(digit)

// }

// return sum === product

// }

W3D1 Unique String

function uniqueString(stringArray) {

// If array has just 1 string return that value

if (stringArray.length === 1) {

return stringArray[0];

}

// Join all array to check if any char is not included in the array

let chars = stringArray.join('');

// Iterate over all characters in combined string

for (ch of chars) {

// Check if any ch of the array contains the ch

let result = stringArray.filter((el) => el.includes(ch));

// Reduce initial array to just filtered 1 unique string which contain

if (result.length === 1) {

return result[0];

}

}

}

W3D2 – Brad the bartender  
  
const maximumCocktails = (recipe,available) => {

// create new var to store total cocktails he can make

let total = [];

// iterate through the ingredient

for (let ingredient in recipe) {

// check if there is enough ingredient in stock for receipe.

if(available[ingredient] >= recipe[ingredient]) {

// Math.floor can be used for rounding

// push into total

total.push(available[ingredient] / recipe[ingredient]) : 0;

} else {

return 0;

}

}

return Math.min(...total);

}

W4D1 - narcissistCNumber

FOR LOOP

function narcissisticNumber(number) {

// Create empty array to store answer

let sum = [];

// Need to convert number to string and seperate

let num = number.toString().split('');

// Loop through num and raise to the power of itself

// console.log(num)

// console.log(num.length)

for(let i=0; i < num.length; i++) {

// Use the num to work out the power of number

// Need to sum that up and check if it is = origanal number

sum.push(num[i] \*\* num.length)

// console.log((num[i] \*\* num.length).reduce((pre, curr) => pre + curr));

}

let reduceVaule = sum.reduce((pre, curr) => pre + curr);

// console.log(sum)

return reduceVaule === number;

}

W4D2 – camelCase

function toCamelCase(str) {

answer = [];

// Convert string to array and ignore space, - or \_

let newArray = str.replace(/\_|-|\\. /g, ' ').split(' ');

// console.log(newArray)

// loop through newArray value and convert to toCamelCase but if input 1st word is capitalise leave it.

for(let i = 1; i < newArray.length; i++) {

// Need to also push substring back to form each word

answer.push(newArray[i].charAt([0]).toUpperCase() + newArray[i].substr(1));

// Join 1st word of array back in

// console.log(answer)

}

return newArray[0] + answer.join('');

}

W5D1 – camelCase

function multiplyValue(value, times){

//return null if 2nd param is not a valid number

if (typeof times !== "number") {

return null;

}

// Check if 1st param is a number. If yes return value \* times

if (typeof value === "number") {

return value \* times;

// if 1st param is a string then use string.repeat to get number of copies of a string

} else if (typeof value === "string") {

return value.repeat(times);

} else {

return null;

}

}

W5D2 – Is it in the list?

function checkList (name, callback) {

fs.readFile('./names.txt', 'utf8', (error, data) => callback(error, data, name))

}

function findName (error, data, name) {

// error clause

if(error) {

console.log(error);

return

}

// split names into array at each new line (/\n/) then use map to convert name to lowecase

let names = data.split(/\n/).map(name => name.toLowerCase());

// make input name lowercase

name = name.toLowerCase();

// make a nicely formatted name for output

capital\_name = name[0].toUpperCase() + name.substring(1)

// output formatted name if in the list

if (names.includes(name)) {

console.log(`${capital\_name} is in the list.`);

} else {

console.log(`${capital\_name} is not in the list.`);

}

}

function findName (error, data, name) {

// error clause

if(error) {

console.log(error);

return

}

// convert names to string and spilt array at each new line (/\n/)

const searchName = data.toString().split(/\n/);

// convert 1st letter of name as uppercase then add substring

const capitalName = name[0].toUpperCase + name.substring(1);

searchName.includes(name) ?

console.log(`${capitalName} is in the list.`) :

console.log(`${capitalName} is not in the list.`);

}

W6D1 – Zodiac sign

function getSign(input) {

// Return 'Invalid Date' if input not a string

if (typeof input !== "string") {

return "Invalid Date"

}

// Convert input to date object as its curr in

const date = new Date(input);

// Convert date to string and split

const [dateNum, month] = date.toLocaleDateString().split('/').map(val => parseInt(val));

// Create switch to return the sign if date is less then or = to

let sign;

switch (month) {

case 1:

sign = dateNum <= 19 ? 'Capricorn' : 'Aquarius';

break;

case 2:

sign = dateNum <= 18 ? 'Aquarius' : 'Pisces';

break;

case 3:

sign = dateNum <= 20 ? 'Pisces' : 'Aries';

break;

case 4:

sign = dateNum <= 19 ? 'Aries' : 'Taurus';

break;

case 5:

sign = dateNum <= 20 ? 'Taurus' : 'Gemini';

break;

case 6:

sign = dateNum <= 20 ? 'Gemini' : 'Cancer';

break;

case 7:

sign = dateNum <= 22 ? 'Cancer' : 'Leo';

break;

case 8:

sign = dateNum <= 22 ? 'Leo' : 'Virgo';

break;

case 9:

sign = dateNum <= 22 ? 'Virgo' : 'Libra';

break;

case 10:

sign = dateNum <= 22 ? 'Libra' : 'Scorpio';

break;

case 11:

sign = dateNum <= 22 ? 'Scorpio' : 'Sagittarius';

break;

case 12:

sign = dateNum <= 21 ? 'Sagittarius' : 'Capricorn';

break;

default:

sign = 'Invalid Date';

break;

}

return sign;

}

W6D2 – Digital Root

// Recursion

const digitalRoot = (n) => {

// Convert n number into strings and split to individual elements

digits = Array.from(String(n), Number);

reduced = digits.reduce((p, c) => p + c);

// If digits value is = 1 return value otherwise call reduce

return digits.length == 1 ? n : digitalRoot(reduced);

}

// While loop

const digitalRoot = (n) => {

// Convert number to string

let nString = n.toString();

while (nString.length > 1) {

nString = Array.from(nString).reduce((p, c) => parseInt(p) + parseInt(c)).toString();

console.log(nString);

}

return parseInt(nString)

}