**Javascript Object Oriented programs**

1.

let dog = {

name: "Spot",

numLegs: 4,

sayLegs: function() {return "This dog has " + this.numLegs + " legs.";}

};

dog.sayLegs();

2.Constructor eg;l

function Dog() {

this.name = "Tommy";

this.color = "Brown";

this.numLegs = 4;

}

function Bird() {

this.name = "Albert";

this.color = "blue";

this.numLegs = 2;

}

let blueBird = new Bird();

3. function Dog(name,color) {

this.name = name;

this.color = color;

this.numLegs = 4;

}

let terrier = new Dog();

4. function House(numBedrooms) {

this.numBedrooms = numBedrooms;

}

// Only change code below this line

let myHouse = new House(4);

myHouse instanceof House;

5. function Bird(name) {

this.name = name;

this.numLegs = 2;

}

let canary = new Bird("Tweety");

let ownProps = [];

// Only change code below this line

for (let property in canary) {

if(canary.hasOwnProperty(property)) {

ownProps.push(property);

}

}

console.log(ownProps);

6. function Dog(name) {

this.name = name;

}

Dog.prototype.numLegs = 4;

// Only change code above this line

let beagle = new Dog("Snoopy");

7. function Dog(name) {

this.name = name;

}

Dog.prototype.numLegs = 4;

let beagle = new Dog("Snoopy");

let ownProps = [];

let prototypeProps = [];

// Only change code below this line

for (let property in beagle) {

if(beagle.hasOwnProperty(property)) {

ownProps.push(property);

} else {

prototypeProps.push(property);

}

}

console.log(ownProps);

console.log(prototypeProps);

8. function Dog(name) {

this.name = name;

}

// Only change code below this line

function joinDogFraternity(candidate) {

if (candidate.constructor === Dog) {

return true;

} else {

return false;

}

}

9, function Dog(name) {

this.name = name;

}

Dog.prototype = {

// Only change code below this line

numLegs: 2,

eat: function() {

console.log("nom nom nom");

},

describe: function() {

console.log("My name is " + this.name);

}

};

10. function Dog(name) {

this.name = name;

}

// Only change code below this line

Dog.prototype = {

constructor: Dog,

numLegs: 4,

eat: function() {

console.log("nom nom nom");

},

describe: function() {

console.log("My name is " + this.name);

}

};

11. function Dog(name) {

  this.name = name;

}

let beagle = new Dog("Snoopy");

// Only change code below this line

Dog.prototype.isPrototypeOf(beagle);

12 function Dog(name) {

  this.name = name;

}

let beagle = new Dog("Snoopy");

Dog.prototype.isPrototypeOf(beagle);  // yields true

// Fix the code below so that it evaluates to true

Object.prototype.isPrototypeOf(Dog.prototype);

13. function Cat(name) {

  this.name = name;

}

Cat.prototype = {

  constructor: Cat,

  };

function Bear(name) {

  this.name = name;

}

Bear.prototype = {

  constructor: Bear,

};

function Animal() { }

Animal.prototype = {

  constructor: Animal,

   eat: function() {

    console.log("nom nom nom");

   }

};

14. function Animal() { }

Animal.prototype = {

  constructor: Animal,

  eat: function() {

    console.log("nom nom nom");

  }

};

// Only change code below this line

let duck = Object.create(Animal.prototype); // Change this line

let beagle = Object.create(Animal.prototype); // Change this line

15 function Animal() { }

Animal.prototype = {

  constructor: Animal,

  eat: function() {

    console.log("nom nom nom");

  }

};

function Dog() { }

// Only change code below this line

//let beagle =  Dog();

Dog.prototype = Object.create(Animal.prototype);

// the above code will inherits the properties of Animal

Dog.eat();// inherits the properties of Animal

16. below child Objects(Dog) creates own methods(bark) and uses parent object method(eat) as well

function Animal() { }

Animal.prototype.eat = function() { console.log("nom nom nom"); };

function Dog() { }

// Only change code below this line

Dog.prototype = Object.create(Animal.prototype);

Dog.prototype.constructor = Dog;

Dog.prototype.bark = function() {

  console.log("Woof!");

};

let beagle = new Dog();

beagle.eat();

beagle.bark();

// Only change code above this line

# 17Override Inherited Methods

1. function Bird() { }
2. Bird.prototype.fly = function() { return "I am flying!"; };
3. function Penguin() { }
4. Penguin.prototype = Object.create(Bird.prototype);
5. Penguin.prototype.constructor = Penguin;
6. // Only change code below this line
7. Penguin.prototype.fly = function() {
8. return "Alas, this is a flightless bird.";
9. };
10. // Only change code above this line
11. let penguin = new Penguin();
12. console.log(penguin.fly());

# Use a Mixin to Add Common Behavior Between Unrelated Objects

1. let bird = {
2. name: "Donald",
3. numLegs: 2
4. };
5. let boat = {
6. name: "Warrior",
7. type: "race-boat"
8. };
9. // Only change code below this line
10. let glideMixin = function(obj) {
11. obj.glide = function() {
12. console.log("Glide ");
13. }
14. };
15. glideMixin(bird);
16. glideMixin(boat);
17. bird.glide();
18. boat.glide();

# Use Closure to Protect Properties Within an Object from Being Modified Externally

In the previous challenge, bird had a public property name. It is considered public because it can be accessed and changed outside of bird's definition.

bird.name = "Duffy";

Therefore, any part of your code can easily change the name of bird to any value. Think about things like passwords and bank accounts being easily changeable by any part of your codebase. That could cause a lot of issues.

The simplest way to make this public property private is by creating a variable within the constructor function. This changes the scope of that variable to be within the constructor function versus available globally. This way, the variable can only be accessed and changed by methods also within the constructor function.

function Bird() {

let hatchedEgg = 10;

this.getHatchedEggCount = function() {

return hatchedEgg;

};

}

let ducky = new Bird();

ducky.getHatchedEggCount();

Here getHatchedEggCount is a privileged method, because it has access to the private variable hatchedEgg. This is possible because hatchedEgg is declared in the same context as getHatchedEggCount. In JavaScript, a function always has access to the context in which it was created. This is called closure.

Eg; function Bird() {

Let weight = 15;

this.getWeight = function(){

return weight;

}

}

# Reset an Inherited Constructor Property

When an object inherits its prototype from another object, it also inherits the supertype's constructor property.

Here's an example:

function Bird() { }

Bird.prototype = Object.create(Animal.prototype);

let duck = new Bird();

duck.constructor

But duck and all instances of Bird should show that they were constructed by Bird and not Animal. To do so, you can manually set the constructor property of Bird to the Bird object:

Bird.prototype.constructor = Bird;

duck.constructor

**Understand the Immediately Invoked Function Expression (IIFE)**

A common pattern in JavaScript is to execute a function as soon as it is declared:

(function () {

console.log("Chirp, chirp!");

})();

This is an anonymous function expression that executes right away, and outputs Chirp, chirp! immediately.

Note that the function has no name and is not stored in a variable. The two parentheses () at the end of the function expression cause it to be immediately executed or invoked. This pattern is known as an immediately invoked function expression or IIFE.

**Use an IIFE to Create a Module**

**An immediately invoked function expression (IIFE) is often used to group related functionality into a single object or module. For example, an earlier challenge defined two mixins:**

**function glideMixin(obj) {**

**obj.glide = function() {**

**console.log("Gliding on the water");**

**};**

**}**

**function flyMixin(obj) {**

**obj.fly = function() {**

**console.log("Flying, wooosh!");**

**};**

**}**

**Note that you have an immediately invoked function expression (IIFE) that returns an object motionModule. This returned object contains all of the mixin behaviors as properties of the object. The advantage of the module pattern is that all of the motion behaviors can be packaged into a single object that can then be used by other parts of your code. Here is an example using it:**

**motionModule.glideMixin(duck);**

**duck.glide();**

# Learn About Functional Programming

Functional programming is a style of programming where solutions are simple, isolated functions, without any side effects outside of the function scope: INPUT -> PROCESS -> OUTPUT

Functional programming is about:

1. Isolated functions - there is no dependence on the state of the program, which includes global variables that are subject to change
2. Pure functions - the same input always gives the same output
3. Functions with limited side effects - any changes, or mutations, to the state of the program outside the function are carefully controlled

Eg;

1. // Function that returns a string representing a cup of green tea
2. const prepareTea = () => 'greenTea';
3. /\*
4. Given a function (representing the tea type) and number of cups needed, the
5. following function returns an array of strings (each representing a cup of
6. a specific type of tea).
7. \*/
8. const getTea = (numOfCups) => {
9. const teaCups = [];
10. for(let cups = 1; cups <= numOfCups; cups += 1) {
11. const teaCup = prepareTea();
12. teaCups.push(teaCup);
13. }
14. return teaCups;
15. };
16. // Only change code below this line
17. const tea4TeamFCC = getTea(40);
18. // Only change code above this line

**2.** // Function that returns a string representing a cup of green tea

const prepareGreenTea = () => 'greenTea';

// Function that returns a string representing a cup of black tea

const prepareBlackTea = () => 'blackTea';

/\*

Given a function (representing the tea type) and number of cups needed, the

following function returns an array of strings (each representing a cup of

a specific type of tea).

\*/

const getTea = (prepareTea, numOfCups) => {

  const teaCups = [];

  for(let cups = 1; cups <= numOfCups; cups += 1) {

    const teaCup = prepareTea();

    teaCups.push(teaCup);

  }

  return teaCups;

};

// Only change code below this line

const tea4GreenTeamFCC = getTea(prepareGreenTea, 27);

const tea4BlackTeamFCC = getTea(prepareBlackTea, 13);

// Only change code above this line

console.log(

  tea4GreenTeamFCC,

  tea4BlackTeamFCC

);

3.

// The global variable

let fixedValue = 4;

function incrementer() {

  // Only change code below this line

     return fixedValue + 1;

  // Only change code above this line

}

let value = incrementer();

console.log(value);

4.

// The global variable

let fixedValue = 4;

// Only change code below this line

function incrementer(arg) {

return arg + 1;

  // Only change code above this line

}

let result = incrementer(fixedValue);

console.log(result);

5. Refactor Global Variables Out of Functions

# Refactor Global Variables Out of Functions

So far, we have seen two distinct principles for functional programming:

1. Don't alter a variable or object - create new variables and objects and return them if need be from a function. Hint: using something like const newArr = arrVar, where arrVar is an array will simply create a reference to the existing variable and not a copy. So changing a value in newArr would change the value in arrVar.
2. Declare function parameters - any computation inside a function depends only on the arguments passed to the function, and not on any global object or variable.

Adding one to a number is not very exciting, but we can apply these principles when working with arrays or more complex objects.

Eg; // The global variable

const bookList = ["The Hound of the Baskervilles", "On The Electrodynamics of Moving Bodies", "Philosophiæ Naturalis Principia Mathematica", "Disquisitiones Arithmeticae"];

// Change code below this line

function add(global\_booklist, bookName) {

      let newlist = [...global\_booklist];

  newlist.push(bookName);

  return newlist;

  // Change code above this line

}

// Change code below this line

function remove(global\_booklist,bookName) {

   let newlist = [...global\_booklist];

  const book\_index = newlist.indexOf(bookName);

  if (book\_index >= 0) {

    newlist.splice(book\_index, 1);

    return newlist;

    // Change code above this line

    }

}

let addBook = add(bookList , "A Brief History of Time");

console.log(addBook);

let removeBook = remove(bookList , "On The Electrodynamics of Moving Bodies");

console.log(removeBook);

**map function**

**eg;**

**// The global variable**

**const watchList = [**

**{**

**"Title": "Inception",**

**"Year": "2010",**

**"Rated": "PG-13",**

**"Released": "16 Jul 2010",**

**"Runtime": "148 min",**

**"Genre": "Action, Adventure, Crime",**

**"Director": "Christopher Nolan",**

**"Writer": "Christopher Nolan",**

**"Actors": "Leonardo DiCaprio, Joseph Gordon-Levitt, Elliot Page, Tom Hardy",**

**"Plot": "A thief, who steals corporate secrets through use of dream-sharing technology, is given the inverse task of planting an idea into the mind of a CEO.",**

**"Language": "English, Japanese, French",**

**"Country": "USA, UK",**

**"Awards": "Won 4 Oscars. Another 143 wins & 198 nominations.",**

**"Poster": "http://ia.media-imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI5OTM0Mw@@.\_V1\_SX300.jpg",**

**"Metascore": "74",**

**"imdbRating": "8.8",**

**"imdbVotes": "1,446,708",**

**"imdbID": "tt1375666",**

**"Type": "movie",**

**"Response": "True"**

**},**

**{**

**"Title": "Interstellar",**

**"Year": "2014",**

**"Rated": "PG-13",**

**"Released": "07 Nov 2014",**

**"Runtime": "169 min",**

**"Genre": "Adventure, Drama, Sci-Fi",**

**"Director": "Christopher Nolan",**

**"Writer": "Jonathan Nolan, Christopher Nolan",**

**"Actors": "Ellen Burstyn, Matthew McConaughey, Mackenzie Foy, John Lithgow",**

**"Plot": "A team of explorers travel through a wormhole in space in an attempt to ensure humanity's survival.",**

**"Language": "English",**

**"Country": "USA, UK",**

**"Awards": "Won 1 Oscar. Another 39 wins & 132 nominations.",**

**"Poster": "http://ia.media-imdb.com/images/M/MV5BMjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM4ODI3MjE@.\_V1\_SX300.jpg",**

**"Metascore": "74",**

**"imdbRating": "8.6",**

**"imdbVotes": "910,366",**

**"imdbID": "tt0816692",**

**"Type": "movie",**

**"Response": "True"**

**},**

**{**

**"Title": "The Dark Knight",**

**"Year": "2008",**

**"Rated": "PG-13",**

**"Released": "18 Jul 2008",**

**"Runtime": "152 min",**

**"Genre": "Action, Adventure, Crime",**

**"Director": "Christopher Nolan",**

**"Writer": "Jonathan Nolan (screenplay), Christopher Nolan (screenplay), Christopher Nolan (story), David S. Goyer (story), Bob Kane (characters)",**

**"Actors": "Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine",**

**"Plot": "When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, the caped crusader must come to terms with one of the greatest psychological tests of his ability to fight injustice.",**

**"Language": "English, Mandarin",**

**"Country": "USA, UK",**

**"Awards": "Won 2 Oscars. Another 146 wins & 142 nominations.",**

**"Poster": "http://ia.media-imdb.com/images/M/MV5BMTMxNTMwODM0NF5BMl5BanBnXkFtZTcwODAyMTk2Mw@@.\_V1\_SX300.jpg",**

**"Metascore": "82",**

**"imdbRating": "9.0",**

**"imdbVotes": "1,652,832",**

**"imdbID": "tt0468569",**

**"Type": "movie",**

**"Response": "True"**

**},**

**{**

**"Title": "Batman Begins",**

**"Year": "2005",**

**"Rated": "PG-13",**

**"Released": "15 Jun 2005",**

**"Runtime": "140 min",**

**"Genre": "Action, Adventure",**

**"Director": "Christopher Nolan",**

**"Writer": "Bob Kane (characters), David S. Goyer (story), Christopher Nolan (screenplay), David S. Goyer (screenplay)",**

**"Actors": "Christian Bale, Michael Caine, Liam Neeson, Katie Holmes",**

**"Plot": "After training with his mentor, Batman begins his fight to free crime-ridden Gotham City from the corruption that Scarecrow and the League of Shadows have cast upon it.",**

**"Language": "English, Urdu, Mandarin",**

**"Country": "USA, UK",**

**"Awards": "Nominated for 1 Oscar. Another 15 wins & 66 nominations.",**

**"Poster": "http://ia.media-imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzUwMTI3.\_V1\_SX300.jpg",**

**"Metascore": "70",**

**"imdbRating": "8.3",**

**"imdbVotes": "972,584",**

**"imdbID": "tt0372784",**

**"Type": "movie",**

**"Response": "True"**

**},**

**{**

**"Title": "Avatar",**

**"Year": "2009",**

**"Rated": "PG-13",**

**"Released": "18 Dec 2009",**

**"Runtime": "162 min",**

**"Genre": "Action, Adventure, Fantasy",**

**"Director": "James Cameron",**

**"Writer": "James Cameron",**

**"Actors": "Sam Worthington, Zoe Saldana, Sigourney Weaver, Stephen Lang",**

**"Plot": "A paraplegic marine dispatched to the moon Pandora on a unique mission becomes torn between following his orders and protecting the world he feels is his home.",**

**"Language": "English, Spanish",**

**"Country": "USA, UK",**

**"Awards": "Won 3 Oscars. Another 80 wins & 121 nominations.",**

**"Poster": "http://ia.media-imdb.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc5MTUwMw@@.\_V1\_SX300.jpg",**

**"Metascore": "83",**

**"imdbRating": "7.9",**

**"imdbVotes": "876,575",**

**"imdbID": "tt0499549",**

**"Type": "movie",**

**"Response": "True"**

**}**

**];**

**map function**

**// Only change code below this line**

**// const ratings = [];**

**// for (let i = 0; i < watchList.length; i++) {**

**// ratings.push({title: watchList[i]["Title"], rating: watchList[i]["imdbRating"]});**

**// }**

**let ratings = [];**

**ratings = watchList.map(film => {**

**let objLists = {};**

**objLists["title"] = film["Title"];**

**objLists["rating"] = film["imdbRating"];**

**return objLists ;**

**})**

**// Only change code above this line**

**console.log(JSON.stringify(ratings));**

**Implement map on a Prototype**

**//global array**

**let s =[23,65,98.5,13];**

**Array.prototype.myMap = function(callback) {**

**const newArray = [];**

**// Only change code below this line**

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

**//let arrElement = this[i];**

**newArray.push(callback(this[i]));**

**}**

**// Only change code above this line**

**return newArray;**

**};**

**let new\_s = s.myMap(function(item)**

**{**

**return item \* 2;**

**});**

**console.log(new\_s);**

**2 Method**

**Array.prototype.myMap = function(callback) {**

**var newArray = [];**

**// Only change code below this line**

**for (let index = 0; index < this.length; index++){**

**//let arrElement = this[index];**

**newArray.push(callback(this[index],index ,this));**

**}**

**// Only change code above this line**

**return newArray;**

**};**

**//global array**

**let inputArr = [23,65,98,5,13]**

**let sqrArry = [];**

**let moveElementlast = [];**

**sqrArry = inputArr.myMap(element => element \* 2);**

**moveElementlast = inputArr.myMap((\_,index,array) => array[index + 1] || array[0]);**

**console.log(sqrArr, moveElementlast);**

**Use the filter Method to Extract Data from an Array**

**// Only change code below this line**

**const filteredList =** watchList**.map(films =>({"title": films['Title'],'rating':films['imdbRating']})).filter(films => films.rating > 8.0)**

**// Only change code above this line**

**console.log(filteredList);**

**Implement the filter Method on a Prototype**

**Array.prototype.myFilter = function(callback) {**

**const newArray = [];**

**// Only change code below this line**

**for (let index = 0; index < this.length; index++){**

**if (callback(this[index],index,this)){**

**newArray.push(this[index]);**

**}**

**}**

**// Only change code above this line**

**return newArray;**

**};**

**console.log([23, 65, 98, 5, 13].myFilter(item => item % 2));**

**console.log([1, 1, 2, 5, 2].myFilter((element, index, array) => array.indexOf(element) === index));**

**OUTPUT**

**[ 23, 65, 5, 13 ]**

**[ 1, 2, 5 ]**

**Return Part of an Array Using the slice Method**

**Note:** **the slice method does not mutates the original array it is called on. Here's an example:**

**function sliceArray(anim, beginSlice, endSlice) {**

**// Only change code below this line**

**let resultArr =[];**

**resultArr = anim.slice(beginSlice , endSlice);**

**return resultArr;**

**// Only change code above this line**

**}**

**const inputAnim = ["Cat", "Dog", "Tiger", "Zebra", "Ant"];**

**let newArr =[];**

**newArr = sliceArray(inputAnim, 1, 3);**

**console.log(newArr);**

**output - [ 'Dog', 'Tiger' ]**

**Remove Elements from an Array Using slice Instead of splice**

**JavaScript offers the splice method for this, which takes arguments for the index of where to start removing items, then the number of items to remove. If the second argument is not provided, the default is to remove items through the end.**

**Note:** **the splice method mutates the original array it is called on. Here's an example:**

**Using the slice method instead of splice helps to avoid any array-mutating side effects.**

**Combine Two Arrays Using the concat Method**

**Concatenation means to join items end to end. JavaScript offers the concat method for both strings and arrays that work in the same way. For arrays, the method is called on one, then another array is provided as the argument to concat, which is added to the end of the first array. It returns a new array and does not mutate either of the original arrays. Here's an example:**

**function nonMutatingConcat(original, attach) {**

**// Only change code below this line**

**let resultArr = original.concat(attach);**

**return resultArr;**

**// Only change code above this line**

**}**

**const first = [1, 2, 3];**

**const second = [4, 5];**

**let myArr = nonMutatingConcat(first, second);**

**console.log(myArr);**

**output-** **[ 1, 2, 3, 4, 5 ]**

**Use the reduce Method to Analyze Data \*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\* important eg;**

**function getRating(inputArry) {**

**// Only change code below this line**

**let filmFilter = inputArry.filter(film => film['Director'] == 'Christopher Nolan');**

**let ratings = filmFilter.map(film => Number(film['imdbRating']));**

**// Note : Number converts strings to numbers**

**let sumRatings = ratings.reduce((sum , rating)=> sum + rating);**

**let averageRating = sumRatings / filmFilter.length ;**

**return averageRating;**

**}**

**console.log(getRating(watchList));**

**OUTPUT: 8.675**

**Use Higher-Order Functions map, filter, or reduce to Solve a Complex Problem**

**const squareList = arr => {**

**// Only change code below this line**

**let intArr = arr.filter( element => (element >= 0 && element % 1 == 0));**

**let sqrResult = intArr.map(element => element \*\* 2);**

**return sqrResult;**

**// Only change code above this line**

**};**

**const squaredIntegers = squareList([-3, 4.8, 5, 3, -3.2]);**

**console.log(squaredIntegers);**

**OUTPUT: [ 25, 9 ]**

**Sort an Array Alphabetically using the sort Method**

**Example 1:**

**Method 1:**

**function ascendingOrder(arr) {**

**return arr.sort(function(a, b) {**

**return a - b;**

**});**

**}**

**ascendingOrder([1, 5, 2, 3, 4]);**

**Output : This would return the value [1, 2, 3, 4, 5].**

**Method 2 using arrow function:**

**function ascendingOrder(arr) {**

**let sorted = arr.sort((a, b) => a - b);**

**return sorted;**

**}**

**ascendingOrder([1, 5, 2, 3, 4]);**

**Output : This would return the value [1, 2, 3, 4, 5].**

**Example 2:**

**function reverseAlpha(arr) {**

**return arr.sort(function(a, b) {**

**return a === b ? 0 : a < b ? 1 : -1; //Desending order a < b**

**});**

**}**

**reverseAlpha(['l', 'h', 'z', 'b', 's']);**

**Output : This would return the value ['z', 's', 'l', 'h', 'b']. // Desending order**

**Example 3:**

**Method 1:**

**function alphabeticalOrder(arr) {**

**return arr.sort(function(a,b){**

**return a === b ? 0 : a > b ? 1 : -1; //Asending order a > b**

**});**

**}**

**console.log(alphabeticalOrder(["a", "d", "c", "a", "z", "g"]));**

**Output : [ 'a', 'a', 'c', 'd', 'g', 'z' ] // Asending order**

**Method 2:using arrow function**

**function alphabeticalOrder(arr) {**

**let sorted = arr.sort( (a,b) => a === b ? 0 : a > b ? 1 : -1);**

**//Asending order a > b**

**return sorted;**

**}**

**console.log(alphabeticalOrder(["a", "d", "c", "a", "z", "g"]));**

**Output : [ 'a', 'a', 'c', 'd', 'g', 'z' ] // Asending order**

**Return a Sorted Array Without Changing the Original Array**

**const globalArray = [5, 6, 3, 2, 9];**

**function nonMutatingSort(arr) {**

**// Only change code below this line**

**let arr2 = arr;**

**let sorted = [].concat(arr2).sort((a, b) => a - b);**

**return sorted;**

**}**

**console.log(nonMutatingSort(globalArray));**

**console.log(globalArray);**

**output**

**[ 2, 3, 5, 6, 9 ]**

**[ 5, 6, 3, 2, 9 ]**

**Split a String into an Array Using the split Method**

**function splitify(str) {**

**// Only change code below this line**

**let str2 = str.split(/[^A-Za-z]/);//Uses Regx**

**return str2 ;**

**// Only change code above this line**

**}**

**console.log(splitify("Hello World,I-am code"));**

**OUTPUT**

**[ 'Hello', 'World', 'I', 'am', 'code' ]**

**Combine an Array into a String Using the join Method**

**function sentensify(str) {**

**// Only change code below this line**

**let str2 = str.split(/[^A-Za-z]/);**

**console.log(str2);**

**let str3 = str2.join(" ");**

**return str3;**

**// Only change code above this line**

**}**

**console.log(sentensify("May-the-force-be-with-you"));**

**OUTPUT**

**[ 'May', 'the', 'force', 'be', 'with', 'you' ]**

**May the force be with you**

**Apply Functional Programming to Convert Strings to URL Slugs**

**// Only change code below this line**

**function urlSlug(title) {**

**let lowerLetters = title.toLowerCase();**

**let splitLetters = lowerLetters.split(' ');**

**//console.log(splitLetters);**

**let filterLetters = splitLetters.filter(element => element !='');// filtering because of extra space in sentences.**

**let hyphenSentence = filterLetters.join('-');**

**return hyphenSentence;**

**}**

**// Only change code above this line**

**console.log("A Mind Needs Books Like A Sword Needs A Whetstone");**

**console.log(urlSlug("A Mind Needs Books Like A Sword Needs A Whetstone"));**

**output**

**A Mind Needs Books Like A Sword Needs A Whetstone**

**a-mind-needs-books-like-a-sword-needs-a-whetstone**

**Use the every Method to Check that Every Element in an Array Meets a Criteria**

**function checkPositive(arr) {**

**// Only change code below this line**

**return arr.every(element => element > 0);**

**// Only change code above this line**

**}**

**console.log(checkPositive([1, 2, 3, -4, 5]));**

**OUTPUT**

**False**

**Use the some Method to Check that Any Elements in an Array Meet a Criteria**

**The some method works with arrays to check if any element passes a particular test. It returns a Boolean value - true if any of the values meet the criteria, false if not.**

**function checkPositive(arr) {**

**// Only change code below this line**

**return arr.some(element => element > 0)**

**// Only change code above this line**

**}**

**console.log(checkPositive([1, 2, 3, -4, 5]));**

**OUTPUT**

**True**

**Introduction to Currying and Partial Application**

**The arity of a function is the number of arguments it requires. Currying a function means to convert a function of N arity into N functions of arity 1.**

**In other words, it restructures a function so it takes one argument, then returns another function that takes the next argument, and so on.**

**Here's an example:**

**function unCurried(x, y) {**

**return x + y;**

**}**

**function curried(x) {**

**return function(y) {**

**return x + y;**

**}**

**}**

**const curried = x => y => x + y**

**curried(1)(2)**

**Example:**

**function add(x) {**

**// Only change code below this line**

**let curried = y => z => x + y + z ;**

**return curried;**

**// Only change code above this line**

**}**

**console.log(add(10)(20)(30));**

**OutPut**

**60**