Functional Programming

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

---

The members of freeCodeCamp happen to love tea.

In the code editor, the `prepareTea` and `getTea` functions are already defined for you. Call the `getTea` function to get 40 cups of tea for the team, and store them in the `tea4TeamFCC` variable.

Solution:

```
/**
 * A long process to prepare tea.
 * @return {string} A cup of tea.
 **/
const prepareTea = () => "greenTea";

/**
 * Get given number of cups of tea.
 * @param {number} numOfCups Number of required cups of t
ea.
 * @return {Array<string>} Given amount of tea cups.
```

```
 **/
const getTea = numOfCups => {
  const teaCups = [];

  for (let cups = 1; cups <= numOfCups; cups += 1) {
    const teaCup = prepareTea();
    teaCups.push(teaCup);
  }

  return teaCups;
};

// Add your code below this line

const tea4TeamFCC = getTea(40); // :(

// Add your code above this line

console.log(tea4TeamFCC);
```

Understand Functional Programming Terminology

The FCC Team had a mood swing and now wants two types of tea: green tea and black tea. General Fact: Client mood swings are pretty common.

With that information, we'll need to revisit the `getTea` function from last challenge to handle various tea requests. We can modify `getTea` to accept a function as a parameter to be able to change the type of tea it prepares. This makes `getTea` more flexible, and gives the programmer more control when client requests change.

But first, let's cover some functional terminology:

Callbacks are the functions that are slipped or passed into another function to decide the invocation of that function. You may have seen them passed to

other methods, for example in `filter`, the callback function tells JavaScript the criteria for how to filter an array.

Functions that can be assigned to a variable, passed into another function, or returned from another function just like any other normal value, are called *first class* functions. In JavaScript, all functions are first class functions.

The functions that take a function as an argument, or return a function as a return value, are called *higher order* functions.

When functions are passed in to or returned from another function, then those functions which were passed in or returned can be called a *lambda*.

Prepare 27 cups of green tea and 13 cups of black tea and store them in `tea4GreenTeamFCC` and `tea4BlackTeamFCC` variables, respectively. Note that the `getTea` function has been modified so it now takes a function as the first argument.

Note: The data (the number of cups of tea) is supplied as the last argument. We'll discuss this more in later lessons.

Solution:

```
// Function that returns a string representing a cup of g
reen tea
const prepareGreenTea = () => 'greenTea';

// Function that returns a string representing a cup of b
lack tea
const prepareBlackTea = () => 'blackTea';

/*
Given a function (representing the tea type) and number o
f cups needed, the
following function returns an array of strings (each repr
esenting 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
);
```

Understand the Hazards of Using Imperative Code

Functional programming is a good habit. It keeps your code easy to manage, and saves you from sneaky bugs. But before we get there, let's look at an imperative approach to programming to highlight where you may have issues.

In English (and many other languages), the imperative tense is used to give commands. Similarly, an imperative style in programming is one that gives the computer a set of statements to perform a task.

Often the statements change the state of the program, like updating global variables. A classic example is writing a `for` loop that gives exact directions to iterate over the indices of an array.

In contrast, functional programming is a form of declarative programming. You tell the computer what you want done by calling a method or function.

JavaScript offers many predefined methods that handle common tasks so you don't need to write out how the computer should perform them. For example, instead of using the `for` loop mentioned above, you could call the `map` method which handles the details of iterating over an array. This helps to avoid semantic errors, like the "Off By One Errors" that were covered in the Debugging section.

Consider the scenario: you are browsing the web in your browser, and want to track the tabs you have opened. Let's try to model this using some simple object-oriented code.

A Window object is made up of tabs, and you usually have more than one Window open. The titles of each open site in each Window object is held in an array. After working in the browser (opening new tabs, merging windows, and closing tabs), you want to print the tabs that are still open. Closed tabs are removed from the array and new tabs (for simplicity) get added to the end of it.

The code editor shows an implementation of this functionality with functions for `tabOpen()`, `tabClose()`, and `join()`. The array `tabs` is part of the Window object that stores the name of the open pages.

Examine the code in the editor. It's using a method that has side effects in the program, causing incorrect behaviour. The final list of open tabs, stored in `finalTabs.tabs`, should be `['FB', 'Gitter', 'Reddit', 'Twitter', 'Medium', 'new tab', 'Netflix', 'YouTube', 'Vine', 'GMail', 'Work mail', 'Docs', 'freeCodeCamp', 'new tab']` but the list produced by the code is slightly different.

Change `Window.prototype.tabClose` so that it removes the correct tab.

Solution:

```
// tabs is an array of titles of each site open within th
e window
const Window = function(tabs) {
  this.tabs = tabs; // We keep a record of the array insi
de the object
};

// When you join two windows into one window
Window.prototype.join = function(otherWindow) {
  this.tabs = this.tabs.concat(otherWindow.tabs);
  return this;
};

// When you open a new tab at the end
Window.prototype.tabOpen = function(tab) {
  this.tabs.push('new tab'); // Let's open a new tab for
now
  return this;
};

// When you close a tab
Window.prototype.tabClose = function(index) {

  // Only change code below this line

  const tabsBeforeIndex = this.tabs.splice(0, index); //
Get the tabs before the tab
  const tabsAfterIndex = this.tabs.splice(1); // Get the
tabs after the tab

  this.tabs = tabsBeforeIndex.concat(tabsAfterIndex); //
Join them together

  // Only change code above this line
```

```
  return this;
 };

// Let's create three browser windows
const workWindow = new Window(['GMail', 'Inbox', 'Work mail', 'Docs', 'freeCodeCamp']); // Your mailbox, drive, and other work sites
const socialWindow = new Window(['FB', 'Gitter', 'Reddit', 'Twitter', 'Medium']); // Social sites
const videoWindow = new Window(['Netflix', 'YouTube', 'Vimeo', 'Vine']); // Entertainment sites

// Now perform the tab opening, closing, and other operations
const finalTabs = socialWindow
  .tabOpen() // Open a new tab for cat memes
  .join(videoWindow.tabClose(2)) // Close third tab in video window, and join
  .join(workWindow.tabClose(1).tabOpen());
console.log(finalTabs.tabs);
```

Avoid Mutations and Side Effects Using Functional Programming

If you haven't already figured it out, the issue in the previous challenge was with the `splice` call in the `tabClose()` function. Unfortunately, `splice` changes the original array it is called on, so the second call to it used a modified array, and gave unexpected results.

This is a small example of a much larger pattern - you call a function on a variable, array, or an object, and the function changes the variable or something in the object.

One of the core principles of functional programming is to not change things. Changes lead to bugs. It's easier to prevent bugs knowing that your functions don't change anything, including the function arguments or any global variable.

The previous example didn't have any complicated operations but the `splice` method changed the original array, and resulted in a bug.

Recall that in functional programming, changing or altering things is called *mutation*, and the outcome is called a *side effect*. A function, ideally, should be a *pure function*, meaning that it does not cause any side effects.

Let's try to master this discipline and not alter any variable or object in our code.

Fill in the code for the function `incrementer` so it returns the value of the global variable `fixedValue` increased by one.

Solution:

```
// the global variable
var fixedValue = 4;

function incrementer() {
  // Add your code below this line
  return fixedValue + 1;

  // Add your code above this line
}

var newValue = incrementer(); // Should equal 5
console.log(fixedValue); // Should print 4
```

Pass Arguments to Avoid External Dependence in a Function

The last challenge was a step closer to functional programming principles, but there is still something missing.

We didn't alter the global variable value, but the function `incrementer` would not work without the global variable `fixedValue` being there.

Another principle of functional programming is to always declare your dependencies explicitly. This means if a function depends on a variable or object being present, then pass that variable or object directly into the function as an argument.

There are several good consequences from this principle. The function is easier to test, you know exactly what input it takes, and it won't depend on anything else in your program.

This can give you more confidence when you alter, remove, or add new code. You would know what you can or cannot change and you can see where the potential traps are.

Finally, the function would always produce the same output for the same set of inputs, no matter what part of the code executes it.

Let's update the `incrementer` function to clearly declare its dependencies.

Write the `incrementer` function so it takes an argument, and then returns a result after increasing the value by one.

Solution:

```
// the global variable
var fixedValue = 4;

// Add your code below this line
function incrementer(value) {
  return value + 1;

  // Add your code above this line
}

var differentValue = incrementer(fixedValue); // Should e
qual 5
console.log(fixedValue); // Should print 4
```

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.

Rewrite the code so the global array `bookList` is not changed inside either function. The `add` function should add the given `bookName` to the end of the array passed to it and return a new array (list). The `remove` function should remove the given `bookName` from the array passed to it.

Note: Both functions should return an array, and any new parameters should be added before the `bookName` parameter.

Solution:

```
// the global variable
var bookList = ["The Hound of the Baskervilles", "On The
Electrodynamics of Moving Bodies", "Philosophiæ Naturalis
 Principia Mathematica", "Disquisitiones Arithmeticae"];

/* This function should add a book to the list and return
 the list */
// New parameters should come before bookName

// Add your code below this line
```

```
function add(arr, bookName) {
  let newArr = [...arr]; // Copy the bookList array to a
new array.
  newArr.push(bookName); // Add bookName parameter to the
 end of the new array.
  return newArr; // Return the new array.
}

/* This function should remove a book from the list and r
eturn the list */
// New parameters should come before the bookName one

// Add your code below this line
function remove(arr, bookName) {
  let newArr = [...arr]; // Copy the bookList array to a
new array.
  if (newArr.indexOf(bookName) >= 0) {
    // Check whether the bookName parameter is in new arr
ay.
    newArr.splice(newArr.indexOf(bookName), 1); // Remove
 the given paramater from the new array.
    return newArr; // Return the new array.
  }
}

var newBookList = add(bookList, 'A Brief History of Time'
);
var newerBookList = remove(bookList, 'On The Electrodynam
ics of Moving Bodies');
var newestBookList = remove(add(bookList, 'A Brief Histor
y of Time'), 'On The Electrodynamics of Moving Bodies');

console.log(bookList);
```

Use the map Method to Extract Data from an Array

So far we have learned to use pure functions to avoid side effects in a program. Also, we have seen the value in having a function only depend on its input arguments.

This is only the beginning. As its name suggests, functional programming is centered around a theory of functions.

It would make sense to be able to pass them as arguments to other functions, and return a function from another function. Functions are considered *first class objects* in JavaScript, which means they can be used like any other object. They can be saved in variables, stored in an object, or passed as function arguments.

Let's start with some simple array functions, which are methods on the array object prototype. In this exercise we are looking at `Array.prototype.map()`, or more simply `map`.

The `map` method iterates over each item in an array and returns a new array containing the results of calling the callback function on each element. It does this without mutating the original array.

When the callback is used, it is passed three arguments. The first argument is the current element being processed. The second is the index of that element and the third is the array upon which the `map` method was called.

See below for an example using the `map` method on the `users` array to return a new array containing only the names of the users as elements. For simplicity, the example only uses the first argument of the callback.

```
const users = [
  { name: 'John', age: 34 },
  { name: 'Amy', age: 20 },
  { name: 'camperCat', age: 10 }
];
```

```
const names = users.map(user => user.name);
console.log(names);
```

The console would display the value `[ 'John', 'Amy', 'camperCat' ]`.

The `watchList` array holds objects with information on several movies.
Use `map` on `watchList` to assign a new array of objects to the `ratings` variable.
Each movie in the new array should have only a `title` key with the name of the film, and a `rating` key with the IMDB rating. The code in the editor currently uses a `for` loop to do this, so you should replace the loop functionality with your `map` expression.

Solution:

```
// 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 throug
h 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 nomin
ations.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI
5OTM0Mw@@._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, Macken
zie Foy, John Lithgow",
    "Plot": "A team of explorers travel through a wormhol
e in space in an attempt to ensure humanity's survival.",
    "Language": "English",
    "Country": "USA, UK",
    "Awards": "Won 1 Oscar. Another 39 wins & 132 nominat
ions.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM
4ODI3MjE@._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 (sto
ry), Christopher Nolan (screenplay), David S. Goyer (scre
enplay)",
    "Actors": "Christian Bale, Michael Caine, Liam Neeson
, Katie Holmes",
    "Plot": "After training with his mentor, Batman begin
s 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 & 6
6 nominations.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzU
wMTI3._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 We
aver, Stephen Lang",
    "Plot": "A paraplegic marine dispatched to the moon P
andora 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 nomina
tions.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc
5MTUwMw@@._V1_SX300.jpg",
    "Metascore": "83",
    "imdbRating": "7.9",
    "imdbVotes": "876,575",
    "imdbID": "tt0499549",
    "Type": "movie",
    "Response": "True"
  }
];

// Only change code below this line

const ratings = watchList.map(item => ({
  title: item["Title"],
  rating: item["imdbRating"]
}));

// Only change code above this line

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

Note: Using ES6 notation, each item in array is processed to extract title and rating.
Parantheses are needed to return an object.

Further Explanation:

This code is using the JavaScript `map()` method to create a new array from the elements of the `watchList` array. The `map()` method applies the provided function to each element of the array and returns a new array with the results.

In this case, the function is creating an object with two properties, `title` and `rating`, and assigning the values of the `Title` and `imdbRating` properties of the current element of the `watchList` array to these properties, respectively. The resulting array, `ratings`, will be an array of objects with a `title` and `rating` properties.

Q.1: What is curly braces in this cose for?

The curly braces `{}` in this code are used to create an object. Inside the curly braces, properties and their corresponding values are defined using key-value pairs.

In this case, the object being created has two properties: `title` and `rating`. The `title` property is being assigned the value of `item["Title"]`, and the `rating` property is being assigned the value of `item["imdbRating"]`. The resulting object is an object with a `title` and `rating` properties.

This is being used inside the map function, where for each element of the watchList array, an object is created with the above properties and returned.

Q.2: What is the parentheses before curly braces for?

The parentheses `()` before the curly braces `{}` in this code are not necessary for the object creation, but they are used to indicate that the object is being returned immediately.

In JavaScript, when you put an object or a function inside parentheses, it is called an "immediately invoked function expression" (IIFE). This is a way to execute a function immediately after it's defined, and it can be used when you want to return a value without assigning it to a variable first.

In this case, the object is not a function and is being returned as it is, so the parentheses do not change the behavior of the code. They are just used to indicate that the object is returned immediately.

Implement map on a Prototype

As you have seen from applying `Array.prototype.map()`, or simply `map()` earlier, the `map` method returns an array of the same length as the one it was called on. It also doesn't alter the original array, as long as its callback function doesn't.

In other words, `map` is a pure function, and its output depends solely on its inputs. Plus, it takes another function as its argument.

You might learn a lot about the `map` method if you implement your own version of it. It is recommended you use a `for` loop or `Array.prototype.forEach()`.

Write your own `Array.prototype.myMap()`, which should behave exactly like `Array.prototype.map()`. You should not use the built-in `map` method. The `Array` instance can be accessed in the `myMap` method using `this`.

Solution:

```
Array.prototype.myMap = function(callback) {
  const newArray = [];
  // Only change code below this line
  for (let i = 0; i < this.length; i++) {
    newArray.push(callback(this[i], i, this));
  }
  // Only change code above this line
  return newArray;
};
```

Use the filter Method to Extract Data from an Array

Another useful array function is `Array.prototype.filter()`, or simply `filter()`.

`filter` calls a function on each element of an array and returns a new array containing only the elements for which that function returns a truthy value - that is, a value which returns `true` if passed to the `Boolean()` constructor. In other words, it filters the array, based on the function passed to it. Like `map`, it does this without needing to modify the original array.

The callback function accepts three arguments. The first argument is the current element being processed. The second is the index of that element and the third is the array upon which the `filter` method was called.

See below for an example using the `filter` method on the `users` array to return a new array containing only the users under the age of 30. For simplicity, the example only uses the first argument of the callback.

```
const users = [
  { name: 'John', age: 34 },
  { name: 'Amy', age: 20 },
  { name: 'camperCat', age: 10 }
];

const usersUnder30 = users.filter(user => user.age < 30);
console.log(usersUnder30);
```

The console would display the value `[ { name: 'Amy', age: 20 }, { name: 'camperCat', age: 10 } ]`.

The variable `watchList` holds an array of objects with information on several movies. Use a combination of `filter` and `map` on `watchList` to assign a new array of objects with only `title` and `rating` keys. The new array should only include objects where `imdbRating` is greater than or equal to 8.0. Note that the `rating` values are saved as strings in the object and you may need to convert them into numbers to perform mathematical operations on them.

Solution:

```
// 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 throug
h 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 nomin
ations.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI
5OTM0Mw@@._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 ha
voc and chaos on the people of Gotham, the caped crusader
 must come to terms with one of the greatest psychologica
l tests of his ability to fight injustice.",
    "Language": "English, Mandarin",
    "Country": "USA, UK",
    "Awards": "Won 2 Oscars. Another 146 wins & 142 nomin
ations.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTMxNTMwODM0NF5BMl5BanBnXkFtZTcwODA
yMTk2Mw@@._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 (sto
ry), Christopher Nolan (screenplay), David S. Goyer (scre
enplay)",
    "Actors": "Christian Bale, Michael Caine, Liam Neeson
, Katie Holmes",
    "Plot": "After training with his mentor, Batman begin
s 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 & 6
6 nominations.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzU
wMTI3._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 We
aver, Stephen Lang",
    "Plot": "A paraplegic marine dispatched to the moon P
andora 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 nomina
tions.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc
5MTUwMw@@._V1_SX300.jpg",
    "Metascore": "83",
```

```
    "imdbRating": "7.9",
    "imdbVotes": "876,575",
    "imdbID": "tt0499549",
    "Type": "movie",
    "Response": "True"
  }
];

// Only change code below this line

const filteredList = watchList
  .filter(movie => {
    // return true it will keep the item
    // return false it will reject the item
    return parseFloat(movie.imdbRating) >= 8.0;
  })
  .map(movie => {
    return {
      title: movie.Title,
      rating: movie.imdbRating
    };
  });

// Only change code above this line

console.log(filteredList);
```

Implement the filter Method on a Prototype

You might learn a lot about the `filter` method if you implement your own version of it. It is recommended you use a `for` loop or `Array.prototype.forEach()`.

Write your own `Array.prototype.myFilter()`, which should behave exactly like `Array.prototype.filter()`. You should not use the built-in `filter` method. The `Array` instance can be accessed in the `myFilter` method using `this`.

Solution:

```
Array.prototype.myFilter = function(callback) {
  const newArray = [];
  // Only change code below this line
  for (let i = 0; i < this.length; i++) {
    if (Boolean(callback(this[i], i, this)) === true) {
      newArray.push(this[i]);
    }
  }
  // Only change code above this line
  return newArray;
};
```

Return Part of an Array Using the slice Method

The `slice` method returns a copy of certain elements of an array. It can take two arguments, the first gives the index of where to begin the slice, the second is the index for where to end the slice (and it's non-inclusive). If the arguments are not provided, the default is to start at the beginning of the array through the end, which is an easy way to make a copy of the entire array.
The `slice` method does not mutate the original array, but returns a new one.

Here's an example:

```
const arr = ["Cat", "Dog", "Tiger", "Zebra"];
const newArray = arr.slice(1, 3);
```

`newArray` would have the value `["Dog", "Tiger"]`.

Use the `slice` method in the `sliceArray` function to return part of the `anim` array given the provided `beginSlice` and `endSlice` indices. The function should return an array.

Solution:

```
function sliceArray(anim, beginSlice, endSlice) {
  // Only change code below this line
return anim.slice(beginSlice, endSlice);
  // Only change code above this line
}

const inputAnim = ["Cat", "Dog", "Tiger", "Zebra", "Ant"]
;
sliceArray(inputAnim, 1, 3);
```

Remove Elements from an Array Using slice Instead of splice

A common pattern while working with arrays is when you want to remove items and keep the rest of the array. 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. However, the `splice` method mutates the original array it is called on. Here's an example:

```
const cities = ["Chicago", "Delhi", "Islamabad", "London",
"Berlin"];
cities.splice(3, 1);
```

Here `splice` returns the string `London` and deletes it from the cities array. `cities` will have the value `["Chicago", "Delhi", "Islamabad", "Berlin"]`.

As we saw in the last challenge, the `slice` method does not mutate the original array, but returns a new one which can be saved into a variable. Recall that

the `slice` method takes two arguments for the indices to begin and end the slice (the end is non-inclusive), and returns those items in a new array. Using the `slice` method instead of `splice` helps to avoid any array-mutating side effects.

Rewrite the function `nonMutatingSplice` by using `slice` instead of `splice`. It should limit the provided `cities` array to a length of 3, and return a new array with only the first three items.

Do not mutate the original array provided to the function.

Solution:

```
function nonMutatingSplice(cities) {
  // Only change code below this line
return cities.slice(0, 3);
  // Only change code above this line
}

const inputCities = ["Chicago", "Delhi", "Islamabad", "Lo
ndon", "Berlin"];
nonMutatingSplice(inputCities);
```

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:

```
[1, 2, 3].concat([4, 5, 6]);
```

The returned array would be `[1, 2, 3, 4, 5, 6]`.

Use the `concat` method in the `nonMutatingConcat` function to concatenate `attach` to the end of `original`. The function should return the concatenated array.

Solution:

```
function nonMutatingConcat(original, attach) {
  // Add your code below this line

  return original.concat(attach);

  // Add your code above this line
}
var first = [1, 2, 3];
var second = [4, 5];
nonMutatingConcat(first, second);
```

Add Elements to the End of an Array Using concat Instead of push

Functional programming is all about creating and using non-mutating functions.

The last challenge introduced the `concat` method as a way to merge arrays into a new array without mutating the original arrays. Compare `concat` to the `push` method. `push` adds items to the end of the same array it is called on, which mutates that array. Here's an example:

```
const arr = [1, 2, 3];
arr.push(4, 5, 6);
```

`arr` would have a modified value of `[1, 2, 3, 4, 5, 6]`, which is not the functional programming way.

`concat` offers a way to merge new items to the end of an array without any mutating side effects.

Change the `nonMutatingPush` function so it uses `concat` to merge `newItem` to the end of `original` without mutating `original` or `newItem` arrays. The function should return an array.

Solution:

Add Elements to the End of an Array Using concat Instead of push

Functional programming is all about creating and using non-mutating functions.

The last challenge introduced the `concat` method as a way to merge arrays into a new array without mutating the original arrays. Compare `concat` to the `push` method. `push` adds items to the end of the same array it is called on, which mutates that array. Here's an example:

```
const arr = [1, 2, 3];
arr.push(4, 5, 6);
```

`arr` would have a modified value of `[1, 2, 3, 4, 5, 6]`, which is not the functional programming way.

`concat` offers a way to merge new items to the end of an array without any mutating side effects.

Change the `nonMutatingPush` function so it uses `concat` to merge `newItem` to the end of `original` without mutating `original` or `newItem` arrays. The function should return an array.

Solution:

```
function nonMutatingPush(original, newItem) {
```

```
  // Add your code below this line

  return original.concat(newItem);

  // Add your code above this line
}

var first = [1, 2, 3];
var second = [4, 5];
nonMutatingPush(first, second);
```

Use the reduce Method to Analyze Data

`Array.prototype.reduce()`, or simply `reduce()`, is the most general of all array operations in JavaScript. You can solve almost any array processing problem using the `reduce` method.

The `reduce` method allows for more general forms of array processing, and it's possible to show that both `filter` and `map` can be derived as special applications of `reduce`. The `reduce` method iterates over each item in an array and returns a single value (i.e. string, number, object, array). This is achieved via a callback function that is called on each iteration.

The callback function accepts four arguments. The first argument is known as the accumulator, which gets assigned the return value of the callback function from the previous iteration, the second is the current element being processed, the third is the index of that element and the fourth is the array upon which `reduce` is called.

In addition to the callback function, `reduce` has an additional parameter which takes an initial value for the accumulator. If this second parameter is not used, then the first iteration is skipped and the second iteration gets passed the first element of the array as the accumulator.

See below for an example using `reduce` on the `users` array to return the sum of all the users' ages. For simplicity, the example only uses the first and second arguments.

```
const users = [
  { name: 'John', age: 34 },
  { name: 'Amy', age: 20 },
  { name: 'camperCat', age: 10 }
];

const sumOfAges = users.reduce((sum, user) => sum + user.age, 0);
console.log(sumOfAges);
```

The console would display the value `64`.

In another example, see how an object can be returned containing the names of the users as properties with their ages as values.

```
const users = [
  { name: 'John', age: 34 },
  { name: 'Amy', age: 20 },
  { name: 'camperCat', age: 10 }
];

const usersObj = users.reduce((obj, user) => {
  obj[user.name] = user.age;
  return obj;
}, {});
console.log(usersObj);
```

The console would display the value `{ John: 34, Amy: 20, camperCat: 10 }`.

The variable `watchList` holds an array of objects with information on several movies. Use `reduce` to find the average IMDB rating of the movies directed by `Christopher Nolan`. Recall from prior challenges how to `filter` data and `map` over it to pull what you need. You may need to create other variables, and return the average rating from `getRating` function. Note that the rating values are saved as strings in the object and need to be converted into numbers before they are used in any mathematical operations.

Solution:

```
// 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 throug
h 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 nomin
ations.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjAxMzY3NjcxNF5BMl5BanBnXkFtZTcwNTI
5OTM0Mw@@._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, Macken
zie Foy, John Lithgow",
    "Plot": "A team of explorers travel through a wormhol
e in space in an attempt to ensure humanity's survival.",
    "Language": "English",
    "Country": "USA, UK",
    "Awards": "Won 1 Oscar. Another 39 wins & 132 nominat
ions.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMjIxNTU4MzY4MF5BMl5BanBnXkFtZTgwMzM
4ODI3MjE@._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 (sto
ry), Christopher Nolan (screenplay), David S. Goyer (scre
enplay)",
    "Actors": "Christian Bale, Michael Caine, Liam Neeson
, Katie Holmes",
    "Plot": "After training with his mentor, Batman begin
s 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 & 6
6 nominations.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BNTM3OTc0MzM2OV5BMl5BanBnXkFtZTYwNzU
wMTI3._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 We
aver, Stephen Lang",
    "Plot": "A paraplegic marine dispatched to the moon P
andora 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 nomina
tions.",
    "Poster": "http://ia.media-
imdb.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc
5MTUwMw@@._V1_SX300.jpg",
    "Metascore": "83",
    "imdbRating": "7.9",
    "imdbVotes": "876,575",
    "imdbID": "tt0499549",
    "Type": "movie",
    "Response": "True"
  }
];

function getRating(watchList){
  // Add your code below this line
  const averageRating = watchList
    // Use filter to find films directed by Christopher N
olan
    .filter(film => film.Director === "Christopher Nolan"
)
    // Use map to convert their ratings from strings to n
umbers
    .map(film => Number(film.imdbRating))
    // Use reduce to add together their ratings
    .reduce((sumOfRatings, rating) => sumOfRatings + rati
ng) /
  // Divide by the number of Nolan films to get the avera
ge rating
  watchList.filter(film => film.Director === "Christopher
 Nolan").length;
  // Add your code above this line
```

```
  return averageRating;
}

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

Notes:

The `reduce()` method in JavaScript applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value. It is used to combine all elements in an array into a single value, or to perform a cumulative operation on the elements of an array.

The main difference between `map()`, `filter()`, and `reduce()` is their purpose:

- `map()` is used to transform an array into a new array of the same size, but with different elements.
- `filter()` is used to select a subset of elements from an array that meet certain criteria, and returns a new array containing only the elements that pass the test.
- `reduce()` is used to combine all elements in an array into a single value, or to perform a cumulative operation on the elements of an array.

In general, you should use `map()` when you want to transform every element in an array, use `filter()` when you want to select a subset of elements that meet certain criteria, and use `reduce()` when you want to combine all elements in an array into a single value or perform a cumulative operation on the elements of an array.

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

Now that you have worked through a few challenges using higher-order functions like `map()`, `filter()`, and `reduce()`, you now get to apply them to solve a more complex challenge.

Complete the code for the `squareList` function using any combination of `map()`, `filter()`, and `reduce()`. The function should return a new array containing the squares of *only* the positive integers (decimal numbers are not

integers) when an array of real numbers is passed to it. An example of an array of real numbers is `[-3, 4.8, 5, 3, -3.2]`.

Note: Your function should not use any kind of `for` or `while` loops or the `forEach()` function.

Solution:

```
const squareList = (arr) => {
  // Only change code below this line
   arr = arr.filter(num => num > 0 && Number.isInteger(nu
m)).map(num => num * num);
  return arr;
  // Only change code above this line
};

const arrayList = [-3, 4.8, 5, 3, -3.2, 0];
const squaredIntegers = squareList(arrayList);
console.log(squaredIntegers)
console.log(arrayList);
```

further explanation:

the code filter first by the condition of > 0 and using Number.isInteger which return true if the element is an Integer then using map to transform each element and return squares of each element into a new array.

Sort an Array Alphabetically using the sort Method

The `sort` method sorts the elements of an array according to the callback function.

For example:

```
function ascendingOrder(arr) {
```

```
  return arr.sort(function(a, b) {
    return a - b;
  });
}

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

This would return the value `[1, 2, 3, 4, 5]`.

```
function reverseAlpha(arr) {
  return arr.sort(function(a, b) {
    return a === b ? 0 : a < b ? 1 : -1;
  });
}

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

This would return the value `['z', 's', 'l', 'h', 'b']`.

JavaScript's default sorting method is by string Unicode point value, which may return unexpected results. Therefore, it is encouraged to provide a callback function to specify how to sort the array items. When such a callback function, normally called `compareFunction`, is supplied, the array elements are sorted according to the return value of the `compareFunction`:
If `compareFunction(a,b)` returns a value less than 0 for two elements `a` and `b`, then `a` will come before `b`. If `compareFunction(a,b)` returns a value greater than 0 for two elements `a` and `b`, then `b` will come before `a`.
If `compareFunction(a,b)` returns a value equal to 0 for two elements `a` and `b`, then `a` and `b` will remain unchanged.

Use the `sort` method in the `alphabeticalOrder` function to sort the elements of `arr` in alphabetical order. The function should return the sorted array.

Solution:

```
function alphabeticalOrder(arr) {
  // Only change code below this line

  return arr.sort(function(a, b) {
    return a === b ? 0 : a > b ? 1 : -1;
  });
  // Only change code above this line
}

alphabeticalOrder(["a", "d", "c", "a", "z", "g"]);
```

further explanation:

In JavaScript, the `sort()` method is used to sort the elements of an array in place and returns the sorted array. By default, the sort() method sorts the array elements in ascending order according to their Unicode code points. However, you can also pass a custom sorting function to the method to sort the array based on specific criteria.

The `sort()` method in JavaScript processes all elements of an array by repeatedly comparing and rearranging elements until the array is sorted. The process is typically implemented as a comparison sort, where the elements are repeatedly compared and swapped until the array is in the desired order.

Here's an example of how the `sort()` method processes all elements of an array:

```
let points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
```

1. First, the element at index 0 (40) is compared to the element at index 1 (100) using the compare function (a-b).
2. Since the result is -60 (40-100), which is negative, it means that 40 is smaller than 100, so 40 is left in its current position.
3. Next, the element at index 1 (100) is compared to the element at index 2 (1) using the compare function.
4. Since the result is 99 (100-1), which is positive, it means that 100 is greater than 1, so the elements at index 1 and 2 switch positions (1,100).
5. The process continues, comparing each element to every other element in the array.
6. When all the comparisons are done, the elements in the array are rearranged in ascending order: [1, 5, 10, 25, 40, 100]

It's worth noting that the implementation of the sort method can vary and the actual sorting algorithm used in JavaScript is implementation dependent, but in any case, it will process all elements of the array to sort it.

Return a Sorted Array Without Changing the Original Array

A side effect of the `sort` method is that it changes the order of the elements in the original array. In other words, it mutates the array in place. One way to avoid this is to first concatenate an empty array to the one being sorted (remember that `slice` and `concat` return a new array), then run the `sort` method.

Use the `sort` method in the `nonMutatingSort` function to sort the elements of an array in ascending order. The function should return a new array, and not mutate the `globalArray` variable.

Solution:

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

function nonMutatingSort(arr) {
  // Only change code below this line
return arr.slice().sort(function(a, b) {
  return a - b;
});

  // Only change code above this line
}

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

Split a String into an Array Using the split Method

The `split` method splits a string into an array of strings. It takes an argument for the delimiter, which can be a character to use to break up the string or a regular expression. For example, if the delimiter is a space, you get an array of words, and if the delimiter is an empty string, you get an array of each character in the string.

Here are two examples that split one string by spaces, then another by digits using a regular expression:

```
const str = "Hello World";
const bySpace = str.split(" ");

const otherString = "How9are7you2today";
const byDigits = otherString.split(/\d/);
```

`bySpace` would have the value `["Hello", "World"]` and `byDigits` would have the value `["How", "are", "you", "today"]`.

Since strings are immutable, the `split` method makes it easier to work with them.

Use the `split` method inside the `splitify` function to split `str` into an array of words. The function should return the array. Note that the words are not always separated by spaces, and the array should not contain punctuation.

Solution:

```
function splitify(str) {
  // Only change code below this line
return str.split(/\W/);

  // Only change code above this line
}
```

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

Combine an Array into a String Using the join Method

The `join` method is used to join the elements of an array together to create a string. It takes an argument for the delimiter that is used to separate the array elements in the string.

Here's an example:

```
const arr = ["Hello", "World"];
const str = arr.join(" ");
```

`str` would have a value of the string `Hello World`.

Use the `join` method (among others) inside the `sentensify` function to make a sentence from the words in the string `str`. The function should return a string. For example, `I-like-Star-Wars` would be converted to `I like Star Wars`. For this challenge, do not use the `replace` method.

Solution:

```
function sentensify(str) {
  // Only change code below this line
return str.split(/\W/).join(" ");
  // Only change code above this line
}

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

Apply Functional Programming to Convert Strings to URL Slugs

The last several challenges covered a number of useful array and string methods that follow functional programming principles. We've also learned about `reduce`, which is a powerful method used to reduce problems to simpler forms. From computing averages to sorting, any array operation can be achieved by applying it. Recall that `map` and `filter` are special cases of `reduce`.

Let's combine what we've learned to solve a practical problem.

Many content management sites (CMS) have the titles of a post added to part of the URL for simple bookmarking purposes. For example, if you write a Medium post titled `Stop Using Reduce`, it's likely the URL would have some form of the title string in it (`.../stop-using-reduce`). You may have already noticed this on the freeCodeCamp site.

Fill in the `urlSlug` function so it converts a string `title` and returns the hyphenated version for the URL. You can use any of the methods covered in this section, and don't use `replace`. Here are the requirements:

The input is a string with spaces and title-cased words

The output is a string with the spaces between words replaced by a hyphen (-)

The output should be all lower-cased letters

The output should not have any spaces

Solution:

```
// Only change code below this line
function urlSlug(title) {
return title
.trim()
.split(/\s+/)
.map(word => word.toLowerCase())
.join("-");


}
```

```
// Only change code above this line

console.log(urlSlug(" Winter Is  Coming"));
```

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

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

For example, the following code would check if every element in the `numbers` array is less than 10:

```
const numbers = [1, 5, 8, 0, 10, 11];


numbers.every(function(currentValue) {
  return currentValue < 10;
});
```

The `every` method would return `false` here.

Use the `every` method inside the `checkPositive` function to check if every element in `arr` is positive. The function should return a Boolean value.

Solution:

```
function checkPositive(arr) {
  // Only change code below this line
return arr.every(value => value > 0);

  // Only change code above this line
}
```

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

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.

For example, the following code would check if any element in the `numbers` array is less than 10:

```
const numbers = [10, 50, 8, 220, 110, 11];

numbers.some(function(currentValue) {
  return currentValue < 10;
});
```

The `some` method would return `true`.

Use the `some` method inside the `checkPositive` function to check if any element in `arr` is positive. The function should return a Boolean value.

Solution:

```
function checkPositive(arr) {
  // Only change code below this line
return arr.some(elem => elem > 0);
  // Only change code above this line
}
```

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

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)
```

`curried(1)(2)` would return `3`.

This is useful in your program if you can't supply all the arguments to a function at one time. You can save each function call into a variable, which will hold the returned function reference that takes the next argument when it's available. Here's an example using the curried function in the example above:

```
const funcForY = curried(1);
console.log(funcForY(2)); // 3
```

Similarly, *partial application* can be described as applying a few arguments to a function at a time and returning another function that is applied to more arguments. Here's an example:

```
function impartial(x, y, z) {
  return x + y + z;
}

const partialFn = impartial.bind(this, 1, 2);
partialFn(10); // 13
```

Fill in the body of the `add` function so it uses currying to add parameters `x`, `y`, and `z`.

Solution:

```
function add(x) {
  // Only change code below this line
return function(y) {
  return function(z) {
    return x + y + z;
  }
};

  // Only change code above this line
}

add(10)(20)(30);
```