script.js

```
// Part 1: Thinking Functionally

// When coding, it is important to approach your work using small, manageable blocks of code.
// Some functions may become dozens or hundreds of lines long, but keeping things small and simple will
// help you scale and maintain your code.
// This section will have you build a few simple functions to accomplish arbitrary tasks.
// When building functions, remember that there are many ways to accomplish a task in programming.
// Sometimes, the shortest route is the best, and sometimes it is not.
// Take the following example, which contains five functions that accomplish the same task.
// If you were looking at this code for the first time, which would make the most sense to you?

// While there is rarely a “correct” answer in programming, it is important to keep your audience (other programmers) in mind. Write functions with descriptive names, and clear inputs and outputs.
// With that in mind, write functions that accomplish the following:


// Take an array of numbers and return the sum.
function arraySum (numbers){
    let sum = 0;
    numbers.forEach(number => {
        sum += number;
    });
    return sum;
}

testArray = [1,5,7,-1];
console.log(arraySum(testArray));


// Take an array of numbers and return the average.
function arrayAverage (numbers){
    let sum = arraySum(numbers);
    return sum/numbers.length;
}
console.log(arrayAverage(testArray));


// Take an array of strings and return the longest string.
function longestString(strings){
    let longestString = '';
    let longestStringLength = 0;
    strings.forEach(string => {
        if (string.length > longestStringLength){
            longestString = string;
            longestStringLength = string.length;
        }
    });
    return longestString;
}

```

```
testStrings = ['hello', 'pie', 'school', 'hi'];
console.log(longestString(testStrings));


// Take an array of strings, and a number and return an array of the strings that are longer than the given number.
function minimumStringLength (strings, length){
    let newStrings = [];
    strings.forEach(string => {
        if (string.length > length){
            if(!newStrings){
                newStrings[0] = string;
            }else{
                newStrings.push(string);
            }
        }
    });
    return newStrings;
}

let newStrings = minimumStringLength(testStrings, 3)
console.log(newStrings);


// For example, stringsLongerThan(['say', 'hello', 'in', 'the', 'morning'], 3); would return ["hello", "morning"].
// Take a number, n, and print every number between 1 and n without using loops. Use recursion.

function printNumbers (number, n){
    if (number <= n){
        console.log(number);
        number++;
        printNumbers(number, n);
    }
}

printNumbers(5,10);









// Part 2: Thinking Methodically

// When functions are built into objects, like Arrays, they are referred to as “methods” of those objects.
// Many methods, including Array methods, require “callback functions” to determine their behavior.
// For the tasks below, use the following data to test your work:
testData = [{ id: "42", name: "Bruce", occupation: "Knight", age: "41" },
 { id: "48", name: "Barry", occupation: "Runner", age: "25" },
 { id: "57", name: "Bob", occupation: "Fry Cook", age: "19" },
 { id: "63", name: "Blaine", occupation: "Quiz Master", age: "58" },
 { id: "7", name: "Bilbo", occupation: "None", age: "111" }]
```

```
// Use callback functions alongside Array methods to accomplish the following:
// Sort the array by age.

function sortAge( p1, p2 ) {return p1.age-p2.age;}
testData.sort(sortAge);
console.log(testData);

// Filter the array to remove entries with an age greater than 50.
function removeAge(people, age){
    return people.filter(person => person.age < age)
}

console.log(removeAge(testData, 50));


// Map the array to change the “occupation” key to “job” and increment every age by 1.

function changeKey (people, orginalKey, newKey){
    const newPeople = people.map((person) => {
        old_key = orginalKey;
        new_key = newKey;

        if (old_key !== new_key) {
            //person[ new_key ] = person[ old_key ];
             Object.defineProperty(person, new_key,
                 Object.getOwnPropertyDescriptor(person, old_key));
            delete person[old_key];
        }

        return person;
      });
      newPeople.forEach (person => {
        //person.age++;
        increaseAge(person); //function created in step3 later on
      });
    return newPeople;
}

console.log(changeKey(testData, 'occupation', 'job'))


// Use the reduce method to calculate the sum of the ages.

let totalAge = Object.values(testData).reduce((t, {age}) =>  t+Number(age), 0);
console.log (totalAge);


// Then use the result to calculate the average age.
let averageAge = totalAge/testData.length;
console.log(averageAge);




// Part 3: Thinking Critically
```

```
158
159 // It is important to remember that when working with objects in JavaScript, we can either pass those objects
160 // into functions by value or by reference. This important distinction changes the way that functions behave,
161 // and can have large impacts on the way a program executes.
162
163 // For this section, develop functions that accomplish the following:
164 // 1) Take an object and increment its age field.
165 function increaseAge (person){
166     checkIfAge(person); // section 3
167     modifiedTime(person); // section 4
168     person.age++;
169 }
170 // 2) Take an object, make a copy, and increment the age field of the copy. Return the copy.
171 function clonePerson (person){
172     let clone = {...person};
173     increaseAge(clone);
174     return clone;
175 }
176
177 // 3) For each of the functions above, if the object does not yet contain an age field, create one and set it to 0.
178 function checkIfAge (person){
179     if (!("age" in person)){
180         person.age = 0;
181     }
182 }
183
184 // 4) Also, add (or modify, as appropriate) an updated_at field that stores a Date object with the current time.
185 function modifiedTime (object){
186     if (!("updated_at" in object)){
187         object.updated_at = 0;
188     }
189     object.updated_at = getTime();
190 }
191
192 function getTime (){
193     let time = {};
194     const date = new Date();
195     time.day = date.getDate();
196     time.month = date.getMonth();
197     time.year = date.getFullYear();
198     time.hours = date.getHours();
199     time.minutes = date.getMinutes();
200     time.seconds = date.getSeconds();
201     return time;
202 }
203
204
205
206 let personA = testData[0];
207 let cloneA = clonePerson(personA);
208
209
210 console.log(personA);
```

```
console.log(cloneA);









//old code









// function part2 (people){
//     const newPeople = people.map(({
//         id,
//         name,
//         occupation: job,
//         ...rest
//       }) => ({
//         id,
//         name,
//         job,
//         ...rest
//       }));
//       newPeople.forEach (person => {
//         //person.age++;
//         increaseAge(person); //function created in step3 later on
//       });
//     return newPeople;
// }

// console.log(part2(testData))
```