**JAVASCRIPT-codecademy**

1. drawName() is a set of repeatable code that we’ve defined elsewhere called a [function](https://www.codecademy.com/resources/docs/general/function" \t "https://www.codecademy.com/courses/learn-what-to-learn/lessons/welcome-to-codecademy/exercises/_blank).

red = [0, 100, 63];

orange = [40, 100, 60];

green = [75, 100, 40];

blue = [196, 77, 55];

purple = [280, 50, 60];

message = 'Change the color!';

drawName(message, blue);

bounceBubbles();

1. There is a new variable named letterColors :
   1. The displayed text in the browser panel will cycle through the values in letterColors in order when drawName() is called with letterColors.

// Color variables:

red = [0, 100, 63];

orange = [40, 100, 60];

green = [75, 100, 40];

blue = [196, 77, 55];

purple = [280, 50, 60];

// Letters in the message will cycle through these colors:

letterColors = [red, orange, green, blue, purple];

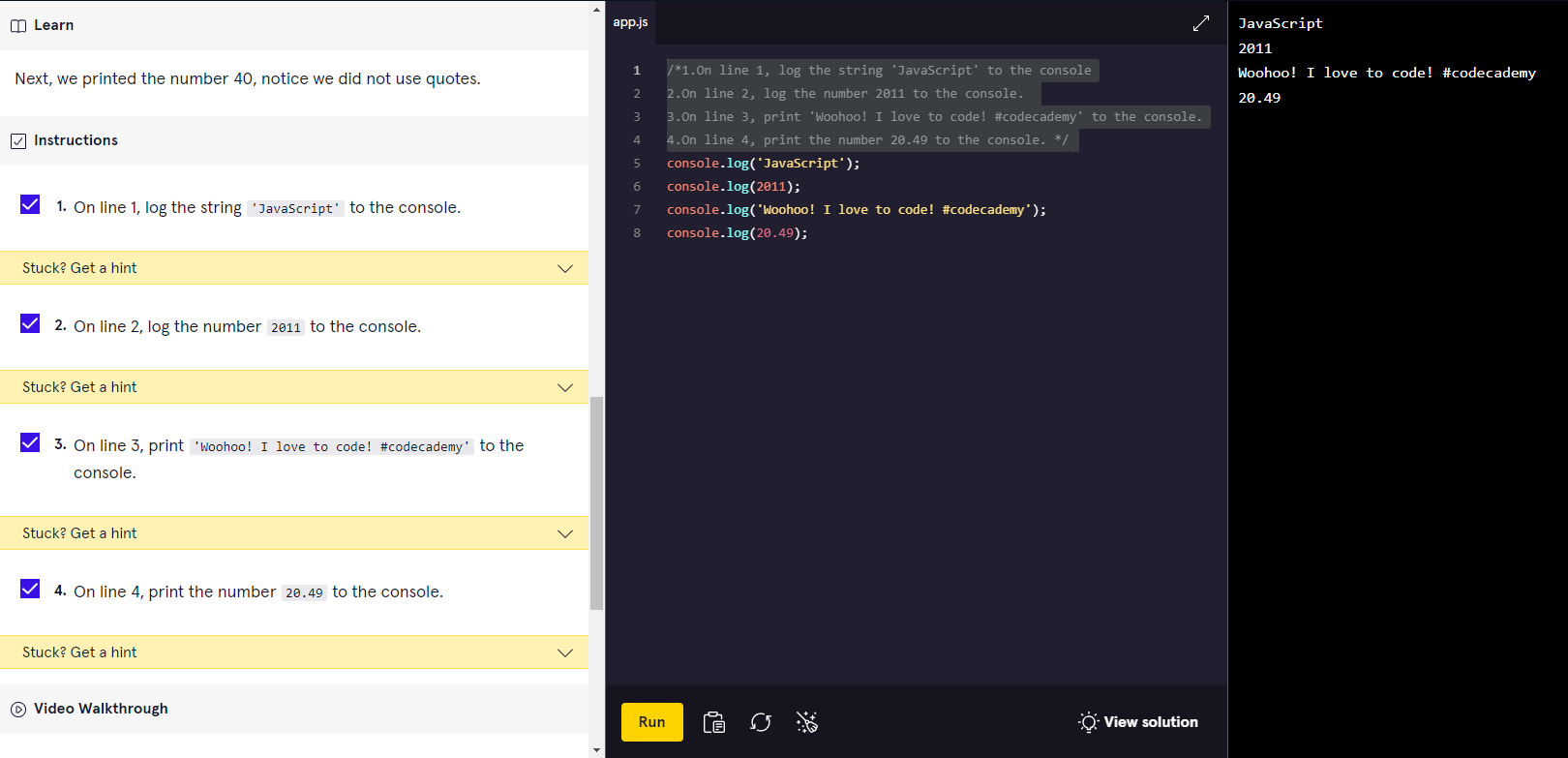
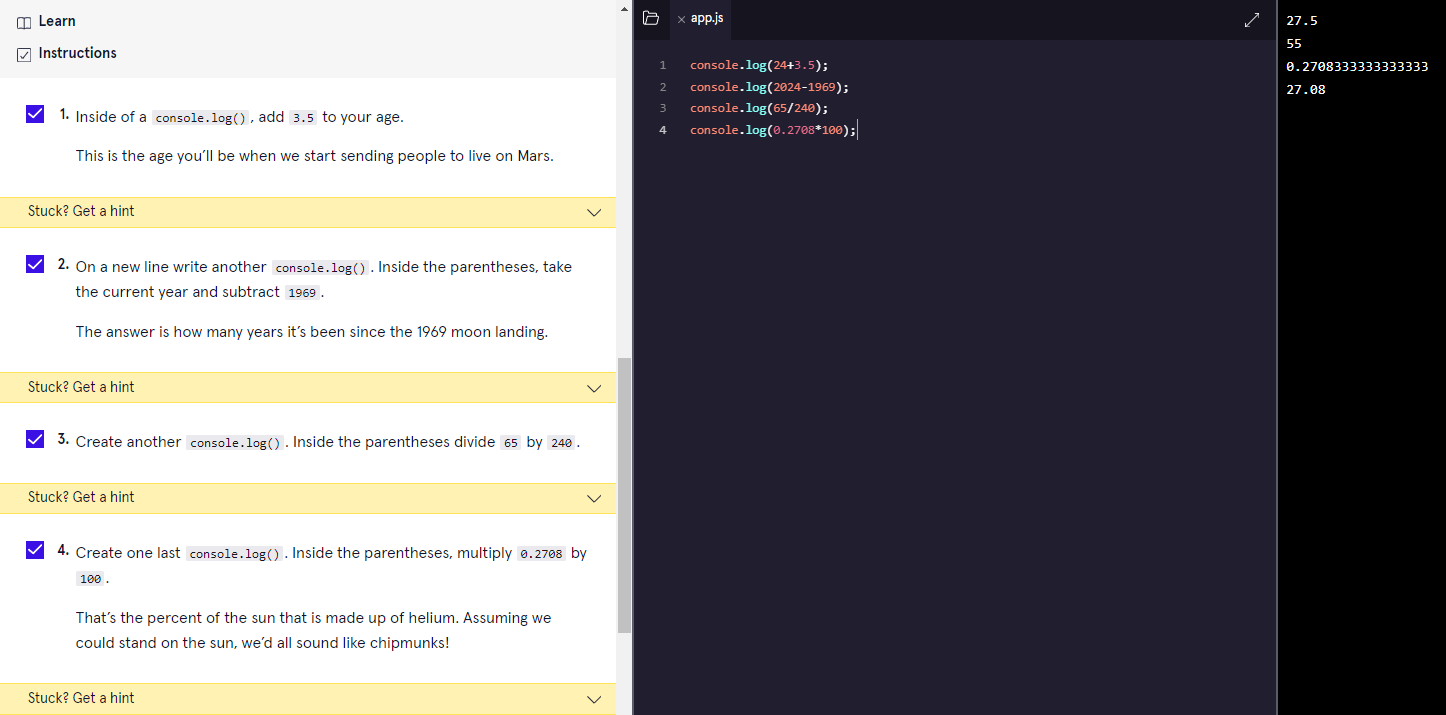
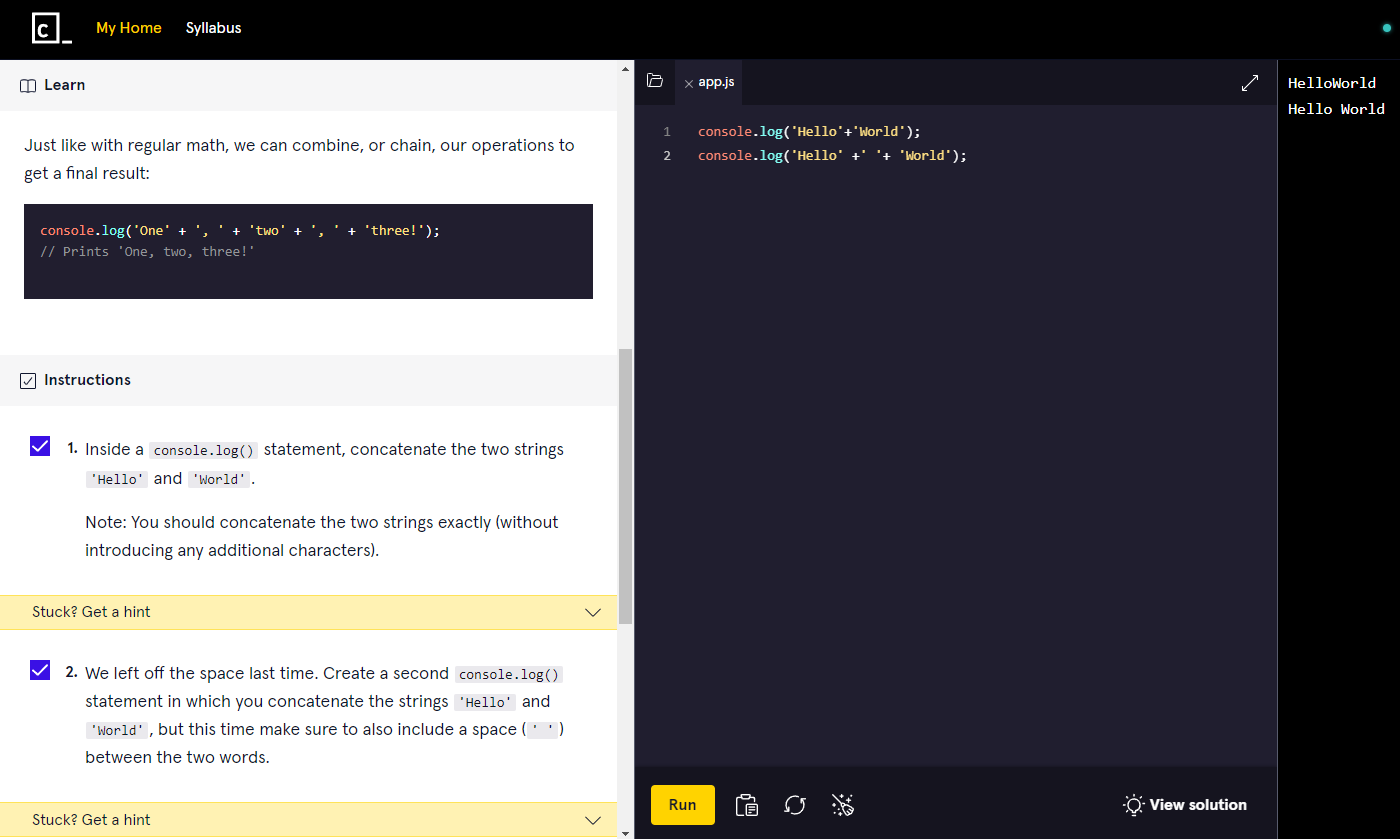
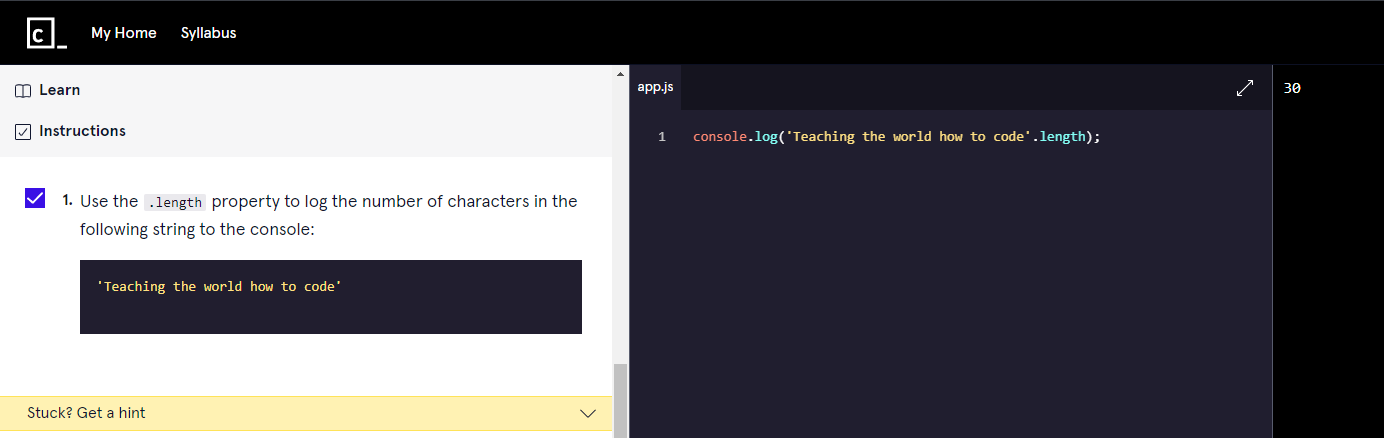
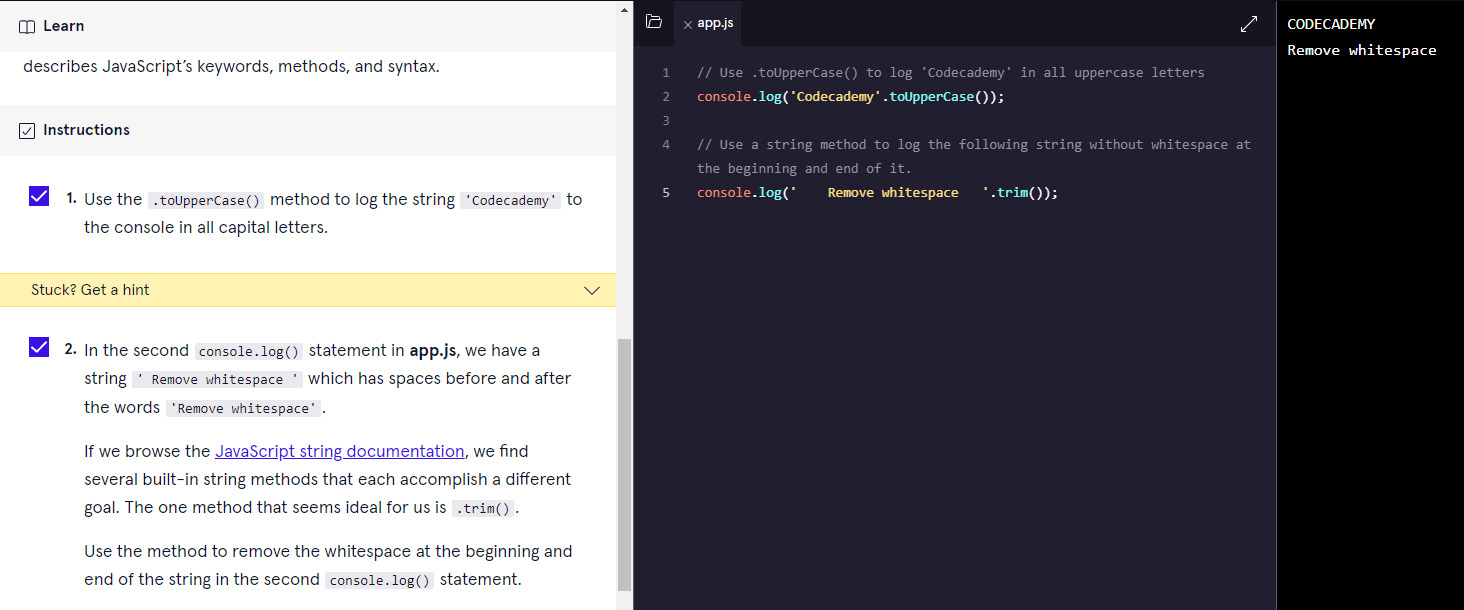
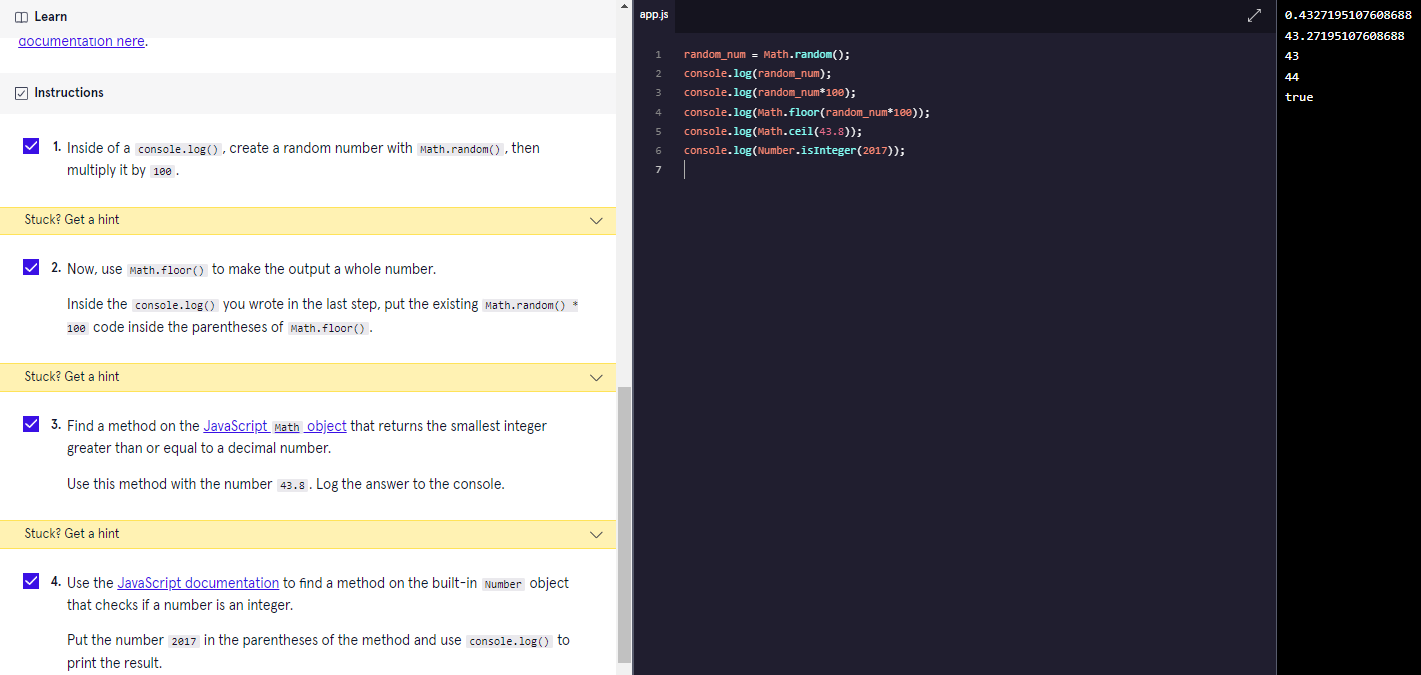
message = 'Multiple colors!';

drawName(message, letterColors);

bounceBubbles();

1. Three variables let you experiment with the animation physics: mouseResponseThreshold, friction, and rotationForce.:
   1. mouseResponseThreshold affects how close the mouse pointer needs to be to affect the dots that make up the letters. The larger the number, the more powerful the effect of the mouse interaction.
   2. You’ll probably want to keep friction between 0 and 1.
   3. rotationForce  represents how much each animated dot will try to rotate when interacting with the mouse.
2. Console:The console is a panel that displays important messages, like [errors](https://www.codecademy.com/resources/docs/javascript/errors" \t "https://www.codecademy.com/courses/introduction-to-javascript/lessons/introduction-to-javascript/exercises/_blank), for developers.
   1. console.log(5); :This example logs 5 to the console.
3. Comments:
   1. Single line comment : console.log(5);  // Prints 5
   2. Multi Line comment :

/\*This is all commented   
console.log(10);  
None of this is going to run!  
console.log(99);  
\*/ .

1. Datatypes:
   1. Number.
   2. BigInt.
   3. String.
   4. Boolean.
   5. Null.
   6. Undefined : undefined means that a given value does not exist
   7. Symbol : symbols are unique identifiers, useful in more complex coding
   8. Object : Collections of related data.
   9. 
2. Arithmetic Operators:
   1. 
3. String Concentration:
   1. 
4. Properties:
   1. 
5. Methods:
   1. 
6. Built-in Objects:
   1. 
7. Variables:
   1. Var.
   2. Let :let keyword signals that the variable can be reassigned a different value.

let changeMe = true;

console.log(changeMe);

changeMe = false;

console.log(changeMe);

* 1. Const :const variable cannot be reassigned because it is constant. If you try to reassign a const variable, you’ll get a TypeError.

const entree = 'Enchiladas';

console.log(entree);

entree = 'Tacos' //TypeError: Assignment to constant variable.

* 1. Mathematical Assignment Operators :

let levelUp = 10;

let powerLevel = 9001;

let multiplyMe = 32;

let quarterMe = 1152;

// Use the mathematical assignments in the space below:

levelUp += 5;

powerLevel -= 100;

multiplyMe \*= 11;

quarterMe /= 4;

// These console.log() statements below will help you check the values of the variables.

// You do not need to edit these statements.

console.log('The value of levelUp:', levelUp);

console.log('The value of powerLevel:', powerLevel);

console.log('The value of multiplyMe:', multiplyMe);

console.log('The value of quarterMe:', quarterMe);

* 1. The Increment and Decrement Operator :

let gainedDollar = 3;

let lostDollar = 50;

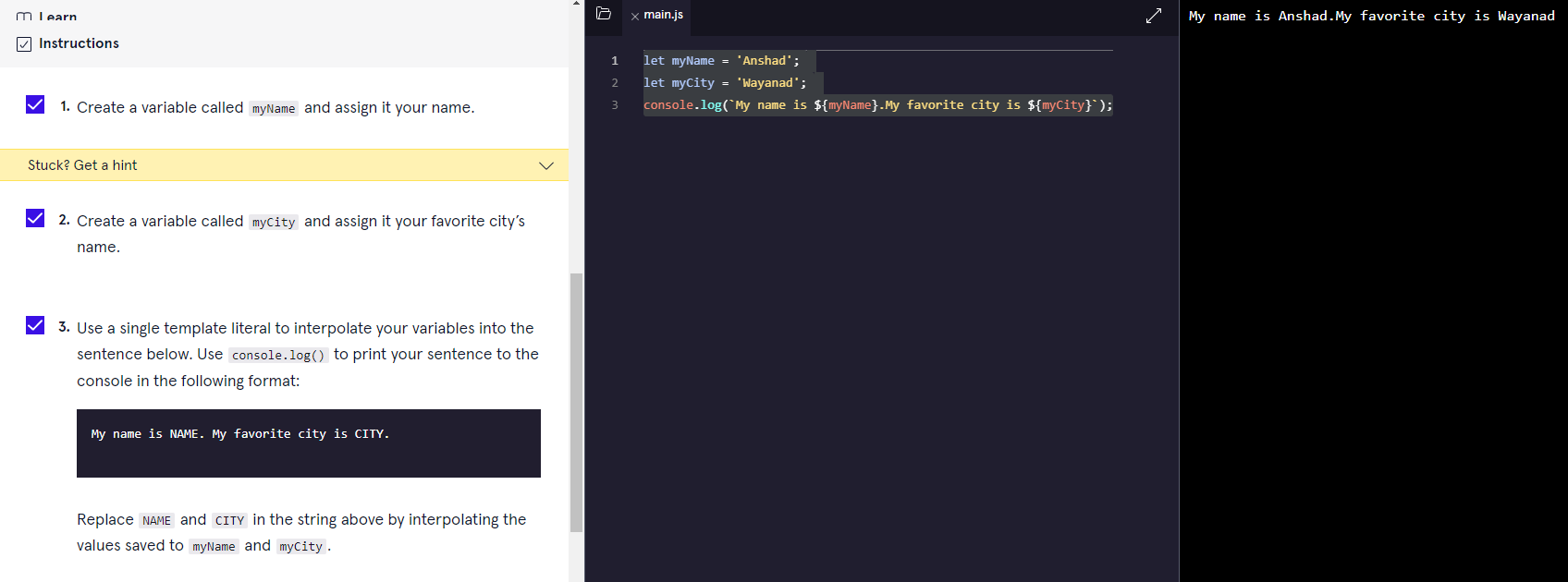
gainedDollar++;

lostDollar--;

* 1. String Concatenation with Variables:

let favoriteAnimal="Lion";

console.log('My favorite animal: '+favoriteAnimal);

* 1. String Interpolation:a template literal is wrapped by backticks ` (this key is usually located on the top of your keyboard, left of the 1 key) :
  2. typeof operator : The typeof operator checks the value to its right and returns, or passes back, a string of the data type.

let newVariable = 'Playing around with typeof.';

console.log(typeof newVariable);

newVariable = 1;

console.log(typeof newVariable);

1. PROJECT-kelvin to Fahrenheit:

//A constant variable 'kelvin' is assigned with a value 293

const kelvin =0;

//variable celsius

const celsius = kelvin-273;

//variable fahrenheit

let fahrenheit = celsius \* (9 / 5) + 32 ;

//getting floor value

fahrenheit = Math.floor(fahrenheit);

console.log(`The temperature is ${fahrenheit} degrees Fahrenheit`);

//convert to Newton

let newton = celsius \* (33 / 100);

newton = Math.round(newton);

console.log(`The temperature is ${newton} degrees Newton`)

1. PROJECT-DOG YEARS:

//declare age as constant

const myAge = 23;

//declare earlyYears

let earlyYears = 2;

earlyYears \*= 10.5;

//declare laterYears

let laterYears = myAge - 2;

laterYears \*= 4;

console.log(earlyYears);

console.log(laterYears);

//add two years

let myAgeDogYears = earlyYears + laterYears;

//lowercase

let myName = 'ANSHAD'.toLowerCase();

console.log(`My name is ${myName}. I am ${myAge} years old in human years which is a ${myAgeDogYears} years old in dog years.`);

1. Conditional Statements:
   1. If statement.

let sale =true;

sale=false;

if (sale) {

  console.log('Time to buy!');

}

* 1. If...Else Statement.

let sale = true;

sale = false;

if(sale) {

  console.log('Time to buy!');

}

else{

  console.log('Time to wait for a sale.');

}

* 1. Comparison Operators.(<,>,<=,>=,===,!==)

let hungerLevel = 7;

if (hungerLevel > 7){

  console.log('Time to eat!');

}

else{

  console.log('We can eat later!');

}

* 1. Logical Operators. (&&,||,!)

let mood = 'sleepy';

let tirednessLevel = 6;

if(mood === 'sleepy' && tirednessLevel > 8){

  console.log('time to sleep');

}

else{

  console.log('not bed time yet');

}

* 1. Truthy and Falsy.

let myVariable = 'I Exist!';

if (myVariable) {

   console.log(myVariable)

} else {

   console.log('The variable does not exist.')

}

* (The code block in the if statement will run because myVariable has a truthy value,even though the value of myVariable is not explicitly the value true).
* **The list of falsy values includes**:

[ 0, Empty strings like "" or '' ,null ,undefined ,NaN ].

let wordCount = 1;

if (wordCount) {

  console.log("Great! You've started your work!");

} else {

  console.log('Better get to work!');

}

let favoritePhrase = '';

if (favoritePhrase) {

  console.log("This string doesn't seem to be empty.");

} else {

  console.log('This string is definitely empty.');

}

* 1. Truthy and Falsy Assignment.
* The code below checks if username is defined and assigns a default string if it is not:

let username = '';

let defaultName;

if (username) {

  defaultName = username;

} else {

  defaultName = 'Stranger';

}

console.log(defaultName);

OR use the || operator :

let username = '';

let defaultName = username || 'Stranger';

console.log(defaultName);

* 1. Ternary Operator.

Example1:

isNightTime ? console.log('TRUE') : console.log('False section');

Example.2:

let favoritePhrase = 'Love That!';

favoritePhrase === 'Love That!'

?

  console.log('I love that!')

:

  console.log("I don't love that!");

* 1. Else If Statements.

let season = 'summer';

if (season === 'spring') {

  console.log('It\'s spring! The trees are budding!');

}

else if(season === 'winter'){

  console.log('It\'s winter! Everything is covered in snow.');

}

else if(season === 'fall'){

  console.log('It\'s fall! Leaves are falling!')

}

else if(season === 'summer'){

  console.log('It\'s sunny and warm because it\'s summer!');

}

else {

  console.log('Invalid season.');

}

* 1. The switch keyword.

let athleteFinalPosition = 'first place';

switch (athleteFinalPosition){

  case 'first place':

    console.log('You get the gold medal!');

    break;

  case 'second place':

    console.log('You get the silver medal!');

    break;

  case 'third place' :

    console.log('You get the bronze medal!');

    break;

  default :

    console.log('No medal awarded.');

    break;

}

* 1. Magic Eight Ball-PROJECT:

let username='';

username ? console.log(`Hello,${username}!`) :

console.log('Hello!');

const userQuestion ='How Many Ball?';

console.log(username+userQuestion);

//We need to generate a random number between 0 and 7.

let randomNumber = Math.floor(Math.random() \* 8);

let eightBall = '';

console.log(randomNumber);

switch(randomNumber){

  case 0:

    eightBall ='It is certain';

    break;

  case 1:

    eightBall ='It is decidedly so';

    break;

  case 2:

    eightBall ='Reply hazy try again';

    break;

  case 3:

    eightBall ='Cannot predict now';

    break;

  case 4:

    eightBall ='Do not count on it';

    break;

  case 5:

    eightBall ='My sources say no';

    break;

  case 6:

    eightBall ='Outlook not so good';

    break;

  case 7:

    eightBall ='Signs point to yes';

    break;

  default:

    console.log('Invalid');

    break;

}

console.log(eightBall);

1. FUNCTIONS in JS:
   1. Declaring a Function:

function getReminder(){

  console.log('Water the plants.');

}

function greetInSpanish(){

  console.log('Buenas tardes');

}

* 1. Calling a Function :

function sayThanks(){

  console.log('Thank you for your purchase! We appreciate your business.');

}

//calling function

sayThanks();

sayThanks();

sayThanks();

* 1. Parameters and Arguments :

function sayThanks(name) {

  console.log('Thank you for your purchase '+ name + '! We appreciate your business.');

}

sayThanks('Anshad');

* 1. Default Parameters :

//default parameters

function makeShoppingList(item1 = 'milk', item2 = 'bread', item3 = 'eggs'){

  console.log(`Remember to buy ${item1}`);

  console.log(`Remember to buy ${item2}`);

  console.log(`Remember to buy ${item3}`);

}

* 1. Return :

function monitorCount(rows , columns){

  return rows \* columns ;

}

const numOfMonitors = monitorCount(5,4);//calls function monitorCount by passing values to it and which returns a result.

console.log(numOfMonitors);

* 1. Helper Functions :

function monitorCount(rows, columns) {

  return rows \* columns;

}

function costOfMonitors(rows , columns){

  return monitorCount(rows,columns) \* 200;

}

const totalCost = costOfMonitors(5,4);

console.log(totalCost);

* 1. Function Expressions :

const plantNeedsWater = function(day){

  if(day === 'Wednesday'){

    return true;

  }

  else {

    return false;

  }

}

console.log(plantNeedsWater('Tuesday'));

* 1. Arrow Functions :
     1. ES6 introduced arrow function syntax, a shorter way to write [functions](https://www.codecademy.com/resources/docs/javascript/functions" \t "https://www.codecademy.com/courses/introduction-to-javascript/lessons/functions/exercises/_blank) by using the special “fat arrow” () => notation.
     2. [Arrow functions](https://www.codecademy.com/resources/docs/javascript/arrow-functions?page_ref=catalog" \t "https://www.codecademy.com/courses/introduction-to-javascript/lessons/functions/exercises/_blank) remove the need to type out the keyword function every time you need to create a function. Instead, you first include the parameters inside the ( ) and then add an arrow => that points to the function body surrounded in { }  Like this:

const rectangleArea = (width, height) => {

  let area = width \* height;

  return area;

};

const plantNeedsWater = (day) => {

  if (day === 'Wednesday') {

    return true;

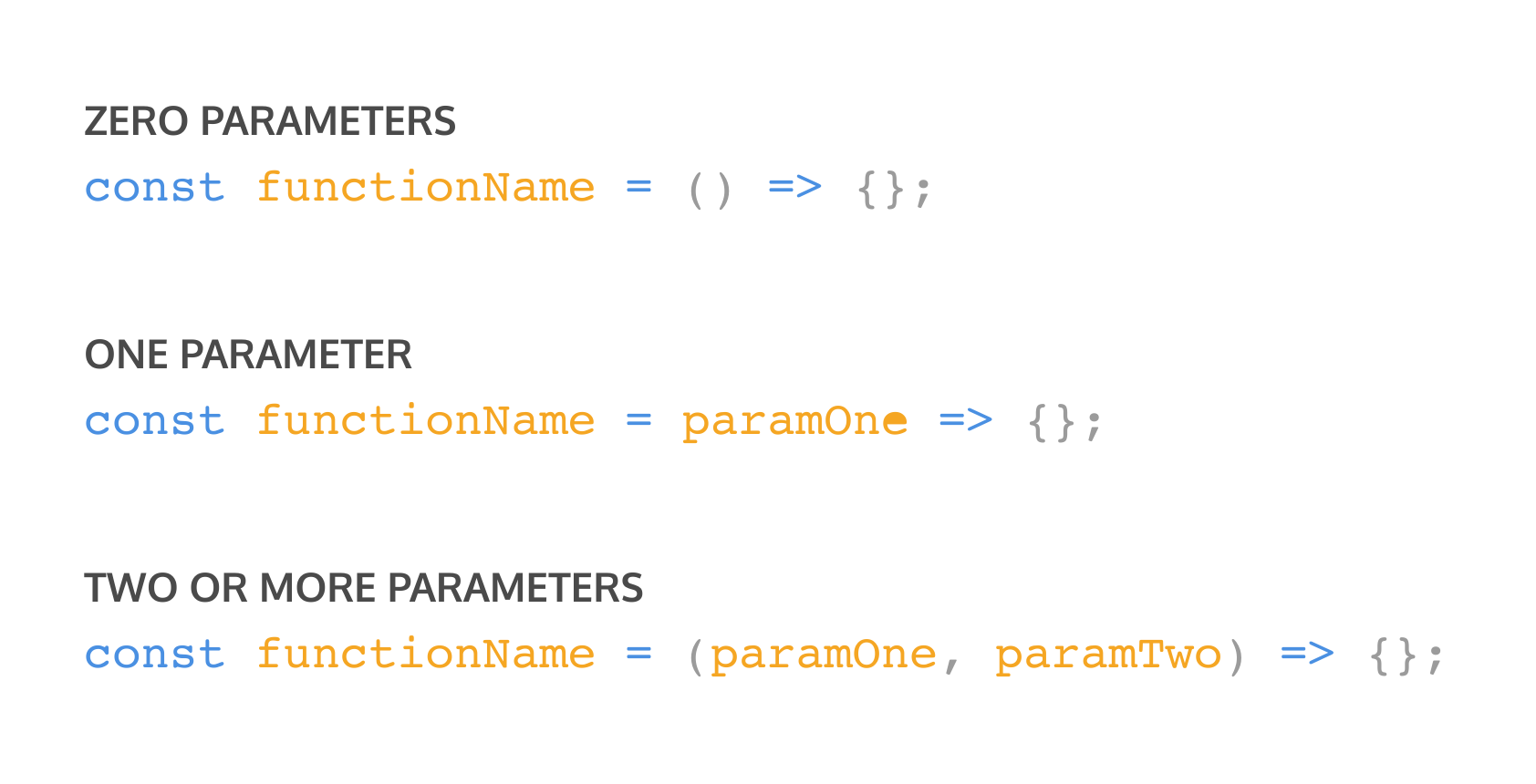
  } else {

    return false;

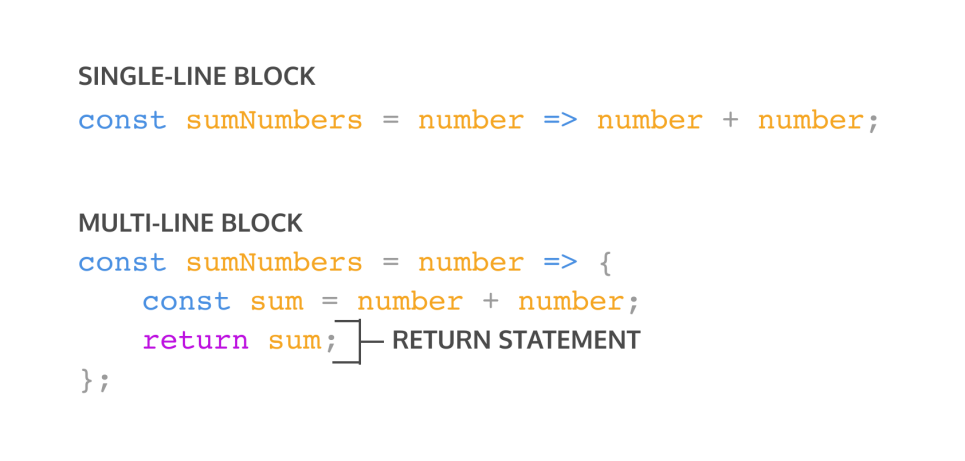
  }

};

* 1. Concise Body Arrow Functions :
     1. JavaScript also provides several ways to refactor [arrow function](https://www.codecademy.com/resources/docs/javascript/arrow-functions?page_ref=catalog" \t "https://www.codecademy.com/courses/introduction-to-javascript/lessons/functions/exercises/_blank) syntax:
        1. Functions that take only a single parameter do not need that parameter to be enclosed in parentheses. However, if a function takes zero or multiple parameters, parentheses are required



* + - 1. A function body composed of a single-line block does not need curly braces.



So if we have a function:

const squareNum = (num) => {  
  return num \* num;  
};

We can refactor the function to:

const squareNum = num => num \* num;

If a function looks like:

const greaterThanFive = (num) => {  
  return num > 5 ? true : false;  
};

To make it into a concise body, we would remove the parentheses, curly braces, and the return keyword:

const greaterThanFive = num =>  num > 5 ? true : false;

1. SCOPE in JS:
   1. Blocks and Scope :

const city = 'New York City'; //Global scope

function logCitySkyline(){

  let skyscraper = 'Empire State Building'; //local Scope

  return 'The stars over the ' + skyscraper + ' in ' + city;

}

console.log(logCitySkyline());

* 1. Global Scope :

let satellite = 'The Moon';

let galaxy = 'The Milky Way';

let stars = 'North Star';

function callMyNightSky(){

  return 'Night Sky: ' + satellite + ', ' + stars + ', and ' + galaxy;

}

console.log(callMyNightSky()) //callMyNightSky() is able to access the global variables

* 1. Block Scope :

function logVisibleLightWaves(){

  const lightWaves = 'Moonlight';

  console.log(lightWaves);

}

logVisibleLightWaves();

console.log(lightWaves); //Gives ReferenceError: lightWaves is not defined

* 1. Scope Pollution :
     1. having too many global variables can cause problems in a program.
     2. Scope pollution is when we have too many global variables that exist in the global namespace, or when we reuse variables across different scopes.
     3. Scope pollution makes it difficult to keep track of our different variables and [sets](https://www.codecademy.com/resources/docs/javascript/sets" \t "https://www.codecademy.com/courses/introduction-to-javascript/lessons/scope/exercises/_blank) us up for potential accidents.
     4. For example, globally scoped variables can collide with other variables that are more locally scoped, causing unexpected behavior in our code.
     5. Let’s look at an example of scope pollution in practice so we know how to avoid it:

let num = 50;  
  
const logNum = () => {  
  num = 100; // Take note of this line of code  
  console.log(num);  
};  
  
logNum(); // Prints 100  
console.log(num); // Prints 100

You’ll notice:

* We have a variable num.
* Inside the function body of logNum(), we want to declare a new variable but forgot to use the let keyword.
* When we call logNum(), num gets reassigned to 100.
* The reassignment inside logNum() affects the global variable num.
* Even though the reassignment is allowed and we won’t get an error, if we decided to use num later, we’ll unknowingly use the new value of num.

const satellite = 'The Moon';

const galaxy = 'The Milky Way';

let stars = 'North Star';

const callMyNightSky = () => {

  stars = 'Sirius';

  return 'Night Sky: ' + satellite + ', ' + stars + ', ' + galaxy;

};

console.log(callMyNightSky());

console.log(stars); //globallly declared stars was reassigned .

* 1. Practice Good Scoping :

const logVisibleLightWaves = () => {

  let lightWaves = 'Moonlight';

  let region = 'The Arctic';

  // Add if statement here:

  if (region === 'The Arctic'){

    let lightWaves = 'Northern Lights';

    console.log(lightWaves);

  }

  console.log(lightWaves);

};

logVisibleLightWaves();

1. Arrays in JS:
   1. Create an Array :

const hobbies = ['Anshad','Nihal','Majo'];

console.log(hobbies);

* 1. Accessing Elements:

const famousSayings = ['Fortune favors the brave.', 'A joke is a very serious thing.', 'Where there is love there is life.'];

let listItem = famousSayings[0];//accessing first element of array

console.log(listItem);

console.log(famousSayings[2]);//accessing third item of array

console.log(famousSayings[3]);//undefined

* 1. Updating Elements :

let groceryList = ['bread', 'tomatoes', 'milk'];

groceryList[1] = 'avocados';

console.log(groceryList);

* 1. Array with let and const :

|  |  |
| --- | --- |
| Code | OutPut |
| let condiments = ['Ketchup', 'Mustard', 'Soy Sauce', 'Sriracha'];  const utensils = ['Fork', 'Knife', 'Chopsticks', 'Spork'];  //re-assign the element in index 0 of condiments to 'Mayo'  condiments[0] = 'Mayo';  console.log(condiments);  //reassign condiments to be a new array that contains a single string ['Mayo']  condiments = ['Mayo']; //arrays declared using 'let' can be reassigned  console.log(condiments);  //Variables declared with the 'const' keyword cannot be reassigned.  //But elements in an array declared with 'const' is mutable.  utensils[3] = 'Spoon';  console.log(utensils); | [ 'Mayo', 'Mustard', 'Soy Sauce', 'Sriracha' ]  [ 'Mayo' ]  [ 'Fork', 'Knife', 'Chopsticks', 'Spoon' ] |

* 1. The .length property :

const objectives = ['Learn a new language', 'Read 52 books', 'Run a marathon'];

console.log(objectives.length);

* 1. The .push() method :

const chores = ['wash dishes', 'do laundry', 'take out trash'];

//To add element to the last of an Array

chores.push('Clean Toilet','Buy Grocerries');

console.log(chores);

* 1. The .pop() method :

const chores = ['wash dishes', 'do laundry', 'take out trash', 'cook dinner', 'mop floor'];

//To remove last element from Array

chores.pop();

console.log(chores);

* 1. More Array methods :

|  |  |
| --- | --- |
| Code | OutPut |
| const groceryList = ['orange juice', 'bananas', 'coffee beans', 'brown rice', 'pasta', 'coconut oil', 'plantains'];  //To remove first element from an Array  groceryList.shift();  console.log(groceryList);  //To add an element to the beginning of an array  groceryList.unshift('popcorn');  console.log(groceryList);  //To get first 3 elements  console.log(groceryList.slice(1,4));  //.slice() is non-mutating!  console.log(groceryList);  //To find the indexOf an element in array:  const pastaIndex = groceryList.indexOf('pasta');  console.log(pastaIndex); | [ 'bananas',    'coffee beans',    'brown rice',    'pasta',    'coconut oil',    'plantains' ]  [ 'popcorn',    'bananas',    'coffee beans',    'brown rice',    'pasta',    'coconut oil',    'plantains' ]  [ 'bananas', 'coffee beans', 'brown rice' ]  [ 'popcorn',    'bananas',    'coffee beans',    'brown rice',    'pasta',    'coconut oil',    'plantains' ]  4 |

* 1. Arrays and Functions :

|  |  |
| --- | --- |
| Code | OutPut |
| const concept = ['arrays', 'can', 'be', 'mutated'];  function changeArr(arr){    arr[3] = 'MUTATED';  }  changeArr(concept);  console.log(concept);  function removeElement(newArr){    newArr.pop();  }  //Here array is passed to a function which then modify the array using .pop()  removeElement(concept);  console.log(concept); | [ 'arrays', 'can', 'be', 'MUTATED' ]  [ 'arrays', 'can', 'be' ] |

* 1. Nested Arrays :

//Nested Array :

let numberClusters = [[1,2],[3,4],[5,6]];

//Accessing Nested Array elements:

const target =  numberClusters[2][1];

console.log(target);

1. LOOPS in JS:
   1. Repeating Task Manually.

|  |  |
| --- | --- |
| Code | OutPut |
| // Write your code below  let vacationSpots = ['Manali','Maldives','Dubai'];  //Displaying array elements without using Loops  console.log(vacationSpots[0]);  console.log(vacationSpots[1]);  console.log(vacationSpots[2]); | Manali  Maldives  Dubai |

* 1. The For Loop :

// Write your code below

for(let i = 5; i <=10 ; i++){

  console.log(i);

}

* 1. Looping in Reverse :

// The loop below loops from 0 to 3. Edit it to loop backwards from 3 to 0

for (let counter = 3; counter >= 0; counter--){

  console.log(counter);

}

* 1. Looping Through Arrays :
  2. Nested Loops :
  3. The While Loop :
  4. Do…While Loop :
  5. The Break Keyword :

1. FUNCTIONS in JS:
2. FUNCTIONS in JS:
3. FUNCTIONS in JS:
4. FUNCTIONS in JS: