Programming with JavaScript Notes

**Writing 1st JavaScript code**

Way to write comments:

/ / This is a one-line comment

/\* This is a multi

Line comment \*/

If you need to ty[e multiple lines of code before you run them, make sure to press SHIFT + ENTER, this shortcut key moves you onto the next line without running the code you’ve just typed up.

Here’s an example of JavaScript code that can be used in the console:

console.log("%cHello, World", "color: blue; font-size: 40px");

Formally known as the “concatenation operator” to output multiple words into the console, you can join them using the + character.

Example:

**console.log("Hello " + "there, " + "World")**

Variables

To assign a variable:

var person = “John”

Now, you’ve taken the value, ‘John’ and assigned it to the ‘person’ variable.

To inspect the contents of the person variable, try:

person;

The ‘console.log method’ can accept one or more values separated by commas.

Example:

Input:

Console.log(“Hello”, person)

Output:

Hello John

Example:

var petDog = "Rex"

var petCat = "Pepper"

console.log(petDog)

console.log(petCat)

console.log("My pet dog's name is", petDog)

console.log("My pet cat's name is", petCat)

var catSound = "purr"

var dogSound = "woof"

console.log(petDog, "says", dogSound)

console.log(petCat, "says", catSound)

catSound = "meow"

console.log(petCat, "now says", catSound)

Data Types

Data: Collective term for the values, or information, within your app.

Primitive Data Types:

1. String
2. Number
3. Boolean
4. Null
5. Undefined
6. BigInt
7. Symbol

Strings & Numbers are essentially values.

Strings are values for description/text.

Numbers are a numerical value.

JavaScript text values are known as string data type.

Boolean data only have two values, True and False.

Null data represents the absence of value.

Undefined data represents a variable that is not yet assigned a value.

BigInt can accommodate a much greater range of numbers than the number data type.

Symbols are used as a unique identifier.

Operators in depth

Logical operators:

* AND: &&
* OR: | |
* NOT: !
* Modulus: %
* Equality: ==
* Strict equality: ===
* Inequality: !=
* Strict inequality: !==
* Addition assignment: +=
* Concatenation assignment: +=

Logical AND operator (in JavaScript):

Is used to confirm if multiple comparisons will return true.

Example: You’re tasked with coming up code that will check if the currentTime variable is between 9 a.m. and 5.p.m. The code needs to console.log true if currentTime > 9 and if currentTime < 17

var currentTime = 10;

console.log(currentTime > 9 && currentTime < 17);

Logical OR operator (in JavaScript):

Is used when you want to check if at least one of the given comparisons evaluates to true.

Example: You’re tasked with writing a program in JavaScript that will return true if the value of the currentTime variable is not between 9 and 17.

var currentTime = 7;

console.log(currentTime < 9 || currentTime > 17);

Logical NOT operator (in JavaScript):

Switches/flips the evaluated Boolean value from true to false and vice versa.

Example:

var petHungry = true;

console.log("Feeding the pet");

console.log("Pet is hungry:", !petHungry);

console.log(petHungry);

To permanently change the value stored in the petHungry variable:

var petHungry = true;

petHungry = !petHungry;

Logical Modulus operator:

Returns the remainder of division.

Example: Imagine a small restaurant that has 4 chairs per table and a total of 5 tables, suddenly receiving 22 guests. How many guests will not be able to sit down?

console.log(22 % 5);

The output is 2, meaning that 2 guests will not be able to sit.

Logical Equality operator:

Checks if two values are equal.

Example:

5 == 5

5 == 6

Output: true, false

Strict Equality operator:

Compares for both values and data types.

Example:

5 === 5

5 === “5”

Output: true, false

Inequality operator:

Checks if two values are not the same.

Example:

5 != “5”

Output: false

Strict Inequality operator:

For this operator to return false, the compared values must have the same value and the same data type.

Example:

5 !== 5

Output: false

+

You can use the + operator on both strings and numbers.

Addition Assignment operator:

Used when one wants to accumulate the values stored in a variable.

Example: You’re counting the number of overtime hours worked in a week. You don’t have to specify the type of work, just the total hours.

You might write a program like this:

var mon = 1;

var tue = 2;

var wed = 1;

var thu = 2;

var fri = 3;

console.log(mon + tue + wed + thu + fr);

Output: 9

You can simplify the above code by using the addition assignment operator:

var overtime = 1;

overtime += 2;

overtime += 1;

overtime += 1;

overtime += 2;

overtime += 3;

console.log(overtime);

Output: 9

Concatenation operator:

This operators syntax is exactly the same of the addition assignment operator, the difference lies in the data type.

Example:

var longString = “”;

longString += “Once”;

longString += “ upon”;

longString += “ a”;

longString += “ time”;

longString += “…”;

console.log(longString);

Output: Once upon a time…

Operator precedence and associativity:

Set of rules that determines what operator should be evaluated first.

Two kinds: left-to-right associativity and right-to-left associativity.

Example: the assignment operator is right-to-left associative, while the greater than operator is left-to-right associative.

var num = 10; // the value on the right is assigned to the variable name on the left

5 > 4 > 3; // The 5 > 4 is evaluated first (to true), then true > 3 is evaluated to false, because the true value is coerced to 1.

Advanced use of operators:

In browser test of knowledge.

1. Using logical && operator

var score = 8;

console.log("Mid-level skills:", score > 0 && score < 10);

1. Using the logical || operator

var timeRemaining = 0;

var energy = 10;

console.log("Game over: ", timeRemaining == 0 || energy == 0)

/ / Note that changing the timeRemaining variable to anything above 0 will change the output

1. Using modulus operator, %, to test if a given number is odd.

var num1 = 2;

var num2 = 5;

var test1 = num1 % 2;

var test2 = num2 % 2;

var result1 = test1 == 0;

var result2 = test2 == 0;

console.log("Is", num1, "an even number?", result1);

console.log("Is", num2, "an even number?", result2);

1. Add numbers using the + operator

console.log(5 + 10)

1. Concatenate numbers and strings using the + operator.

var now = "Now in ";

var three = 3;

var d = "D!";

console.log(now + three + d);

1. Use the += operator to accumulate values in a variable.

var counter = 0;

counter += 5;

counter += 3;

console.log(counter)

JavaScript improvements

In 1996, the European Computer Manufacturers Association (ECMA) made a deal with Netscape) (The company who tasked Brenan Eich to essentially create LiveScript, renamed to JavaScript after it rose to power) to draft the specification of JavaScript, and in 1997 – the first edition of ECMAScript was published.

ECMA published this specification to set the standard, calling it the ECMA-262 standard.

JavaScript as a language is not a standalone entity, it only exists as an implementation known as JavaScript engines.

The JavaScript engine itself comes with different ways to interact with various parts of the browser, these are known as Browser API’s.

UNTIL, Ryan Dahl built Node.js, coming up with a way that uses a JavaScript engine as a standalone entity. Now it’s possible to use JavaScript outside the browser, as a separate program on the command line, or as a server side environment.

Today, JavaScript is ubiquitous and is running on any device that can run a JavaScript engine.

Additional resources:

JavaScript primitive values:

<https://developer.mozilla.org/en-US/docs/Glossary/Primitive>

iQuery Official Website:

<https://jquery.com/>

StackOverflow Developer Survey 2021 Most Popular Technologies:  
<https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies>

ECMA262 Specification:

<https://tc39.es/ecma262/>

Conditional Statements Example

“if else” and “switch” statements are used to determine the program execution flow based on whether or not some conditions have been met.

This is why they are sometimes referred to as flow control statements.

Generally, if else statements are better suited if there is a binary choice in the condition.

When there are smaller number of possible outcomes of truthy checks, it is still possible to use an if else statement, such as:

if(light == "green") {

console.log("Drive")

} else if (light == "orange") {

console.log("Get ready")

} else if (light == "red") {

console.log("Dont' drive")

} else {

//this block will run if no condition matches

console.log("The car is not green, orange, or red");

}

However, if there are a lot of possible outcomes, it is best practice to use a switch statement because it is easier less verbose. Being easier to read, it is easier to follow the logic, and thus reduce cognitive load of reading multiple conditions.

//converting the previous if-else example with switch-case

switch(light) {

case 'green':

console.log("Drive");

break;

case 'orange':

console.log("Get ready");

break;

case 'red':

console.log("Don't drive");

break;

default:

//this block will run if no condition matches

console.log('The light is not green, orange, or red');

break;

}

Practice Conditional Statements:

Are you old enough?

var age = 10;

if (age >= 65) {

console.log('You get your income from your pension')

} else if (age < 65 && age >= 18) {

console.log('Each month you get a salary')

} else if (age < 18) {

console.log('You get an allowance')

} else {

console.log('The value of the age variable is not numerical')

}

Code the days of the week program as a switch statement.

var day = "Sunday";

switch(day) {

case 'Monday':

console.log("Take the week!");

break;

case 'Tuesday':

console.log("Still got it!");

break;

case 'Wednesday':

console.log("Half way there.");

break

case 'Thursday':

console.log("One more day after today!");

break;

case 'Friday':

console.log("It's Friday!!");

break;

case 'Saturday':

console.log("Enjoy the weekend.");

break;

case 'Sunday':

console.log("The holy day.");

break;

default:

console.log("There is no such day.");

break;

}

Repetitive tasks with loops

1. Write a for loop that will perform exactly the same repetitive code as this:

console.log(1)

console.log(2)

console.log(3)

console.log(4)

console.log(5)

console.log('Counting completed!')

for (let i = 1; i <= 5; i++) {

console.log(i);

}

console.log("Counting completed");

1. Write a for loops that will perform exactly the same repetitive code as this:

console.log(5)

console.log(4)

console.log(3)

console.log(2)

console.log(1)

console.log('Countdown finished!')

for (let i = 5; i >= 1; i--) {

console.log(i);

}

console.log("Counting completed");

1. Write a while loop that will perform exactly the same repetitive code as this:

console.log(1)

console.log(2)

console.log(3)

console.log(4)

console.log(5)

console.log('Counting completed!')

var i = 1;

while (i <= 5) {

console.log(i);

i++;

}

console.log("Counting Completed")

1. Write a while loop that will perform exactly the same repetitive code as this:

console.log(5)

console.log(4)

console.log(3)

console.log(2)

console.log(1)

console.log('Countdown finished!')

var i = 5;

while (i >= 1) {

console.log(i);

i--;

}

console.log("Counting Completed")

1. Write a while loop that will perform exactly the same repetitive code as this:

console.log(2018)

console.log(2019)

console.log(2020)

console.log(2021)

console.log(2022)

var i = 2018;

while (i <= 2022) {

console.log(i);

i++;

}

Loops and Nested Loops

//single loop

for (var firstNum = 0; firstNum < 2; firstNum++) {  
 console.log(firstNum);  
}

Output:

0

1

By changing the value of of 2 in “firstNum < 2”, you can alter how long the loop goes on. You can also nest loops inside one another, like so:

for (var firstNum = 0; firstNum < 2; firstNum++) {

for (var secondNum = 0; secondNum < 10; secondNum++) {

console.log(firstNum + ", " + secondNum);

}

}

Output:

0, 0

0, 1

0, 2

…

Now that there’s a list of all the numbers that will be multiplied, having the actual result of this multiplication is as easy as updating the console.log() call.

for (var firstNum = 0; firstNum < 2; firstNum++) {

for (var secondNum = 0; secondNum < 10; secondNum++) {

console.log(firstNum + " times " + secondNum + " equals " + firstNum \* secondNum);

}

}

Output:

0 times 0 equals 0

…

Another example would be making a custom division table:

for (var i = 100; i > 10; i = i - 10) {

for (var j = 10; j > 4; j = j - 5) {

console.log(i + " divided by " + j + " equals " + i / j);

}

}

Uses of Loops

Loops can be used to display grids

var cubes = 'ABCDEFG';

//styling console output using CSS with a %c format specifier

for (var i = 0; i < cubes.length; i++) {

var styles = "font-size: 40px; border-radius: 10px; border: 1px solid blue; background: pink; color: purple";

console.log("%c" + cubes[i], styles)

}

cubes.length returns a number, since cubes are defined as a string of characters, it gives the length of the string saved as the variable. (therefore it’s interchangeable with 7)

cubes[i] simply targets each individual letter in the loop, based on the current value of the i variable. (when i == 0, it’s A. when i == 1, it’s B)

Loops are incredibly important, used often, and very versatile. Here are some more examples:

* Looping over blog posts in some structured data, and displaying each blog post title on a blog home page.
* Looping over social media posts in some structured data, and displaying each social media post based on some conditions.
* Looping over some structured data on clothing available for sale in an online clothing store, and displaying relevant data for each item of clothing.

Working with Conditionals and Loops

Exercise: Using a for loop, create a list that goes from 1-10 but alternates the first three numbers as, gold, silver, and bronze medals. Use an if-else statement, then try with a switch statement.

for (var i = 1; i <= 10; i++) {

if ( i == 1) {

console.log("Gold medal")

} else if (i == 2) {

console.log("Silver medal")

} else if (i == 3) {

console.log("Bronze medal")

} else {

console.log(i)

}

}

for (var i = 1; i <= 10; i++) {

switch(i) {

case 1:

console.log("Gold medal");

break;

case 2:

console.log("Silver medal");

break;

case 3:

console.log("Bronze medal");

break;

default:

console.log(i);

break;

}

}

**Arrays, Objects, and Functions**

Building and Calling Functions

Start by giving the function declaration a name:

function listArrayItems(arr) {

// ... code to be added ...

}

Now, listArrayItems has been declared a function. It’s setup to accept a single parameter, arr (which stands for an array).

Now I’ll code a for loop to loop over the array.

The loop needs to:

1. Start the loop counter as a temporary variable (i).
2. The exit condition (the maximum value of the loop counter variable (i), above which the loop no longer runs).
3. How to update the value of (i) after each loop.

Additional info:

1. Loop counter will be set to 0 because arrays are counted from 0.
2. This results in a one-to-one mapping of the current value of (i) at any given time, corresponding to the same index position of any item in the arr array 2.
3. The for loops exit condition is when the value of (i) is equal or greater than arr.length
4. Since arr.length counts the number of items in the array from one, and the array items are indexed from zero, this effectively means that as soon as (i) is equal to arr.length, the loop will finish. And any other code after it will be run.
5. This practically means that the exit condition for this loop will be i < arr.length returning false.
6. In other words, as long as i < arr.length is true, the loop will continue to run. To make sure none of the items in the array are skipped, I have to increase the value of (i) by 1 after each loop.

function listArrayItems(arr) {

for (var i = 0; i < arr.length; i++) {

// ... code pending here ...

}

}

All that’s left is deciding how you want to output each item from the received arr array.

It can be as simple as console logging the array item of the current value of (i).

function listArrayItems(arr) {

for (var i = 0; i < arr.length; i++) {

// ... code pending here ...

}

}

Now invoke the listArrayItems function, for example, give it the following array of colors:

var colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple', 'pink'];

listArrayItems(colors); //display all items in the array at once

This will display all the items in the array at once.

To have these displayed with the number in front of each item, you can add this code to the previous bit. (Must add (i) to the console log).

//function that takes an array as input and display all items of this array

function listArrayItems(arr) {

for (var i = 0; i < arr.length; i++) {

console.log(i, arr[i])

}

}

var colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple', 'pink'];

listArrayItems(colors);

To start the count from one instead of zero, you can update the declaration as follows:

function listArrayItems(arr) {

for (var i = 0; i < arr.length; i++) {

console.log(i+1, arr[i])

}

}

Here, you simply added a “+1” to the (i) in the console log.

You can add control flow right inside the function, in this example it’s based on whether a specific array member matches a specific value – in this case, the string ‘red’.

function listArrayItems(arr) {

for (var i = 0; i < arr.length; i++) {

if (arr[i] == 'red') {

console.log(i\*100, "tomato!")

} else {

console.log(i\*100, arr[i])

}

}

}

var colors = ['red', 'orange', 'yellow', 'green', 'blue', 'purple', 'pink'];

listArrayItems(colors);

Practicing with Functions

Task 1:

Write a function named letterFinder that accepts two parameters: *word* and *match*.

function letterFinder(word, match) {

}

Task 2:

Code a ‘for loop’ inside the functions body. The loops counter should start at zero, increment by 1 on each iteration and exit when the counters variable’s equal to the length of the word parameter

function letterFinder(word, match) {

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

//this loop exists when i is equal to the length of the word

}

}

Task 3:

Add an if statement inside the loop whose conditions work as follows:

1. Access each of the letters inside the passed in *word* using the counter variable, with *word[i]*.
2. Check if the current *word[i]* is equal to the value of *match*.

function letterFinder(word, match) {

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

if(word[i] == match) {

//check if the current characater, word[i], is equal to the match

}

}

}

Task 4:  
Console log the following inside the body of the if statement:

Console.log(‘Found the’, match, ‘at’, i)

function letterFinder(word, match) {

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

if(word[i] == match) {

//check if the current characater, word[i], is equal to the match

console.log('Found the', match, 'at', i)

}

}

}

Task 5:  
Write the else condition. Here you’ll just console log the following:

Console.log(‘---No match found at,’ i)

function letterFinder(word, match) {

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

if(word[i] == match) {

//check if the current characater, word[i], is equal to the match

console.log('Found the', match, 'at', i)

} else {

console.log('---No match found at', i)

}

}

}

Task 6:

Call the letterFinder and pass it as its first argument as the string “test” and its second argument, the string “t”.

function letterFinder(word, match) {

for(var i = 0; i < word.length; i++) {

if(word[i] == match) {

//check if the current characater, word[i], is equal to the match

console.log('Found the', match, 'at', i)

} else {

console.log('---No match found at', i)

}

}

}

letterFinder("test", "t")

Object Literals and the Dot Notation

One of the most common ways of building an object in JavaScript is using the object literal syntax: {}  
To be able to access this object literal, it is very common to assign it to a variable, such as:

var user = {}; //create an object

An object literal assigned to a variable means that the object is bound to be extended and manipulated in a myriad of ways.

Sometimes, an entire object can be immediately built, using the object literal syntax, by specifying the object’s properties, delimited as key value pairs.

//creating an object with properties and their values

var assistantManager = {

rangeTilesPerTurn: 3,

socialSkills: 30,

streetSmarts: 30,

health: 40,

specialAbility: "young and ambitious",

greeting: "Let's make some money"

}

This syntax is very easy to read, essentially consisting of two steps:

1. Declaring a new variable and assigning an object literal to it – in other words, this: var assistantManager = {}
2. Assigning the values to each of the object’s keys, using the assignment operator, =

It’s very easy to build any kind of object in JavaScript using this syntax.

For example, here’s a table object:

var table = {

legs: 3,

color: "brown",

priceUSD: 100,

}

To access the table object, I can simply console log the entire object:

console.log(table);//display the object in the developer console

The returned value is the entire table object:

{legs: 3, color: 'brown', priceUSD: 100}

Additionally, you can console log any individual property, like this:

console.log(table.color); // 'brown'

With this syntax recipe, you can build any other object in a similar way:

var house = {

rooms: 3,

color: "brown",

priceUSD: 10000,

}

An alternative approach of building objects is to first save an empty object literal to a variable, then use the dot notation to declare new properties. Use the assignment operator to add values to those properties, for example:

var house2 = {};

house2.rooms = 4;

house2.color = "pink";

house2.priceUSD = 12345;

Additionally, nothing is preventing me from combining the two approaches. For example:

console.log(house); // {rooms: 3, color: "brown", priceUSD: 10000}

house.windows = 10;

console.log(house); // {rooms: 3, color: "brown", priceUSD: 10000, windows: 10}

This flexibility additionally means that you can update already existing properties, not just add new ones:

house.windows = 11;

console.log(house); // {rooms: 3, color: "brown", priceUSD: 10000, windows: 11}

Object Literals and the Brackets Notation

An alternative to the dot notation, is the brackets notation.

Here, you can see an example (from Object Literals and Dot Notation), this time using the brackets notation:

var house2 = {};

house2["rooms"] = 4;

house2['color']= "pink";

house2["priceUSD"] = 12345;

console.log(house2); // {rooms: 4, color: 'pink', priceUSD: 12345}

By using the brackets notation, you’re essentially wrapping each property’s key as a **string**, inside either the single or double quotes – just like with regular strings.

Then the entire property key is wrapped into an opening and a closing square bracket.

You can access and update properties on objects using either the dot notation, or the brackets notation, or a combination of both. Here’s an example:

var car = {};

car.color = "red";

car["color"] = "green";

car["speed"] = 200;

car.speed = 100;

console.log(car); // {color: "green", speed: 100}

With the brackets notation, you can add space characters inside the property names, like so:

car["number of doors"] = 4;

console.log(car); // {color: 'green', speed: 100, number of doors: 5}

Additionally, you can add numbers (as the string data type) as property keys:

car["2022"] = 1901;

console.log(car); // {2022: 1901, color: 'green', speed: 100, number of doors: 5}

Though this is discouraged, having a property key as a numbers string is not really conveying useful information.

The brackets notation can also evaluate expressions. Here’s an example:

var arrOfKeys = ['speed', 'altitude', 'color'];

var drone = {

speed: 100,

altitude: 200,

color: "red"

}

for (var i = 0; i < arrOfKeys.length; i++) {

console.log(drone[arrOfKeys[i]])

}

This will result in the output:

100

200

red

Using the fact that brackets can evaluate expressions, I accessed the arrOfKeys[i] property on the drone object.

While the for loop was running, the value was changed each loop.

Specifically, the first time it ran it was evaluated like this:

* The value of i was 0
* The value of arrOfKeys[i] was arrOfKeys[0], which was “speed”
* Thus, drone[arrOfKeys[i]] was evaluated to drone[“speed”] which is equal to 100

This allowed me to loop over each of the values stored inside the drone object, based on each of its properties’ keys.

Arrays are Objects

In JavaScript, arrays are objects. Which means arrays also have some built-in properties and methods.

One of the most commonly used built-in methods on arrays are the push() and the pop() methods.

To add new items to an array, you can use the push() method:

var fruits = [];

fruits.push("apple"); // ['apple']

fruits.push('pear'); // ['apple', 'pear']

To remove the last item from an array, you can use the pop() method:

fruits.pop();

console.log(fruits); // ['apple']

You can now build a function that takes all its arguments and pushes them into an array, like this:

function arrayBuilder(one, two, three) {

var arr = [];

arr.push(one);

arr.push(two);

arr.push(three);

console.log(arr);

}

You can now call the arrayBuilder() function, for example:

arrayBuilder('apple', 'pear', 'plum'); // ['apple', 'pear', 'plum']

Even better, you don’t have to console log the newly built array.

Instead, you can return it:

function arrayBuilder(one, two, three) {

var arr = [];

arr.push(one);

arr.push(two);

arr.push(three);

return arr;

}

Additionally, you can save this function to a variable.

You can name it anything but for this example, I’ll use the name simpleArr:

var simpleArr = arrayBuilder('apple', 'pear', 'plum');

And now you can console log the values stored in simpleArr:

console.log(simpleArr); // ['apple','pear','plum']

Math and Object Cheat Sheet

JavaScript has many built-in objects. One of which is the Math object.

Here are some of the built-in methods in the Math object:

Number Constants:

* The PI number: Math.PI which is approximately 3.14159.
* The Euler’s constant: Math.E which is approximately 2.718.
* The natural logarithm of 2: Math.LN2 which is approximately 0.693.

Rounding methods:

* Math.ceil() – rounds up to the closest integer.
* Math.floor() – rounds down to the closest integer.
* Math.round() – rounds up/down to the closest integer.
* Math.trunc() – trims the decimal, leaving only the integer.

Arithmetic and calculus methods:

* Math.pow(2, 3) – Calculates the number 2 to the power of 3, results in 8.
* Math.sqrt(16) – calculates the square root of 16, results in 4.
* Math.cbrt(8) – finds the cube root of 8, results in 2.
* Math.abs(-10) – returns the absolute value, result is 10.
* Logarithmic methods: Math.log(), Math.log2(), Math.log10()
* Return the minimum and maximum values of all the inputs: Math.min(9, 8, 7) returns 7. Math.max(9, 8, 7) returns 9.
* Trigonometric methods: Math.sin(), Math.cos(), Math.tan(), etc.

String Cheat Sheet

For most of the examples in this section I will be using these variables:

var greet = "Hello, ";

var place = "World"

All strings have several built-in properties at their disposal. The property length is extremely useful, here’s an example of its usage:

greet.length; // 7

To read each character at a specific index in a string, starting from zero, I can use the charAt() method:

greet.charAt(0); // 'H'

The concat() method joins the two strings:

"Wo".concat("rl").concat("d"); // 'World'

The indexOf returns the location of the first position that matches a character.

"ho-ho-ho".indexOf('h'); // 0

"ho-ho-ho".indexOf('o'); // 1

"ho-ho-ho".indexOf('-'); // 2

The lastIndexOf finds the last match, otherwise it works the same as indexOf.

The split method chops up the string into an array of sub-strings:

"ho-ho-ho".split("-"); // ['ho', 'ho', 'ho']

There are also some methods to change the casing of strings, for example:

greet.toUpperCase(); // "HELLO, "

greet.toLowerCase(); // "hello, "

Here’s a list of all the methods that were just covered:

* charAt()
* concat()
* indexOf()
* lastIndexOf()
* split()
* toUpperCase()
* toLowerCase()

Exercise: Creating Arrays and Objects

1. Create a new empty array literal and assign it to the variable clothes.
2. Add 5 of your favorite items of clothing as strings using the push() method.
3. Remove the fifth piece of clothing from the array using the pop() method.
4. Add a new piece of clothing using the push().
5. Use console.log to show the third item from the clothes array in the console.
6. Create a new empty object literal and assign it to the variable favCar.
7. Using the dot notation, assign a color property to the favCar object and give it a string value with the color of your choice.
8. Using the dot notation, assign a convertible property to the favCar object and give it a Boolean value of your choice.
9. Use the console to log the entire favCar object

var clothes = [];

clothes.push("Hat");

clothes.push("Watch");

clothes.push("Sweatshirt");

clothes.push("Glasses");

clothes.push("Shoes");

clothes.pop()

clothes.push("T")

console.log(clothes[2])

var favCar = {};

favCar.color = "black";

favCar.convertible = false;

console.log(favCar)

Object Methods

It’s important to remember just how many properties can be included in objects, like functions.

If the function is property of an object, it is then referred to as a method.

var car = {};

car.color = "red";

//add a method to the car object so that it can be called as car.turnkey()

car.turnKey = function() {

console.log('engine running');

}

Here’s a demonstration, creating an object using something known as the constructor function.

var car = {};

car.mileage = 98765;

car.color = "red";

console.log(car);

Now that I have an object with some properties I will add a method. This method, when called, will output some text to the console.

Keep in mind, a method is just another property of an object.

In this case, what’s unique is that the value assigned is a function.

All-in-all, all the key-value pairs in an object are referred to simply as properties. However, if I want to refer to the properties that can be executed, then I refer to them as methods.

var car = {};

car.mileage = 98765;

car.color = "red";

console.log(car);

car.turnTheKey = function() {

console.log("The engine is running")

}

console.log(car);

Now I want to add another method, lightsOn.

var car = {};

car.mileage = 98765;

car.color = "red";

console.log(car);

car.turnTheKey = function() {

console.log("The engine is running")

}

car.lightsOn = function() {

console.log("The lights are on.")

}

console.log(car);

car.turnTheKey();

car.lightsOn()

Now that I have an object with a couple properties, two of which that are methods, I need to try executing the turnTheKey and the lightsOn methods.

This method can be accessed only through objects. So to run it, it should call oon the object and then the method, using the dot notation.

**Error Handling**

Syntax, Logical, and Runtime errors

Some of the most common errors in JavaScript:

* Reference Error
* Syntax Error
* Type Error
* Range Error

Other errors include:

* Aggregate Errors
* Error
* Internal Error
* URI Error

ReferenceError

A ReferenceError gets thrown when, for example, one tries to use variables that haven’t been declared anywhere.

An example of a Refeence Error would be when you called on a variable that doesn’t exist

console.log(username);

Uncaught ReferenceError: username is not defined;

SyntaxError

Any kind of invalid JavaScript code will cause a SyntaxError

TypeError

A TypeError is thrown when, for example, trying to run a method on a non-supoorted data type.

A simple example is attempting to run the pop() method on a string:

"hello".pop() // Uncaught TypeError: "hello".pop is not a function

Strings do have all array the array methods readily available to them, and trying to use some of those methods will result in a TypeError being thrown.

RangeError

A RangeError is thrown when we’re giving a value to a function, but that value is out of the allowed range of acceptable input values.

This is an example of converting a Base 10 number to base 2 number. (common decimal system to binary number)

(10).toString(2); // '1010'

(10).toString(100); // Uncaught RangeError: toString() radix argument must be between 2 and 36

Exercise: Error Prevention

Instructions:

Task 1: Code a function declaration

You need to code a function declaration named addTwoNums, which accepts numbers a and b and console logs a + b.

Task 2: Invoke the addTwoNums with a number and a string.

Invoke the addTwoNums using the following arguments: 5 and “5”.

Task 3: Update the addTwoNums function with a try…catch block

Add the try and catch blocks inside the function definitions body. For now, just make sure that the console log of a + bis inside the try block. Additionally, the catch block should catch an error named err and, inside the body of the catch block, you need to console log the err value.

Task 4: If the passed-in arguments are not numbers, throw an error.

If either of the arguments passed to the addTwoNums are not numbers, you’ll throw an error.

Specifically, code a condition with the following logic:

* If the typeof the a parameter is not equal to ‘number’, throw a new ReferenceError. Inside the ReferenceError, pass a custom error message of ‘the first argument is not a number’.
* Else if the typeof the b parameter is not equal to ‘number’, throw a new ReferenceError. Inside the ReferenceError, pass a custom error merssage of ‘the second argument is not a number’.
* Else, console log a + b

Once completed, all the code inside the try block will be inside these conditional statements.

Task 5: Update the catch block

Inside the catch block, update the code from console.log(err) to console.log(“Error!”, err).

Task 6: Invoke the addTwoNums function

Invoke the addTwoNums function using 5 and “5” as arguments.

Task 7: Add another console log under the addTwoNums function invocation.

Add another line of code that console logs the string “It still works”

1: Code a function declaration.

function addTwoNums(a, b) {

console.log(a + b)

}

2: Invoke the addTwoNums function with a number and a string.

function addTwoNums(a, b) {

console.log(a + b)

}

addTwoNums(5, "5")

//55

3: Update the addTwoNums function with a try…catch block.

function addTwoNums(a, b) {

try {

console.log(a + b);

} catch (err) {

console.log(err)

}

}

addTwoNums(5, "5")

4: If the passed-in arguments are not numbers, throw an error.

function addTwoNums(a,b) {

try {

if(typeof(a) != 'number') {

throw new ReferenceError('the first argument is not a number')

} else if (typeof(b) != 'number') {

throw new ReferenceError('the second argument is not a number')

} else {

console.log(a + b)

}

} catch(err) {

console.log(err)

}

}

5: Update the catch block

function addTwoNums(a,b) {

try {

if(typeof(a) != 'number') {

throw new ReferenceError('the first argument is not a number')

} else if (typeof(b) != 'number') {

throw new ReferenceError('the second argument is not a number')

} else {

console.log(a + b)

}

} catch(err) {

console.log("Error!", err)

}

}

6 & 7: Invoke the addTwoNums function & Add another console log under the addTwoNums function invocation.

function addTwoNums(a,b) {

try {

if(typeof(a) != 'number') {

throw new ReferenceError('the first argument is not a number')

} else if (typeof(b) != 'number') {

throw new ReferenceError('the second argument is not a number')

} else {

console.log(a + b)

}

} catch(err) {

console.log("Error!", err)

}

}

addTwoNums(5, "5")

console.log("It still works")

Exercise: Defensive Programming

Defensive programming is all about assuming that all the arguments a function will receive are of the wrong type, the wrong value or both.

In summary, you are assuming that things will go wrong and you are proactive in thinking about such scenarios before they happen, so as to make your function less likely to cause errors because of faulty inputs.

In this exercise, I’m ensuring that both arguments that are passed in will satisfy the following criteria:

* The length of the word parameter cannot be less than 2.
* The length of the match parameter must be 1.
* The type of both the word and the match parameters must be string.

You will use the code below to complete the task:

function letterFinder(word, match) {

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

if(word[i] == match) {

//if the current character at position i in the word is equal to the match

console.log('Found the', match, 'at', i)

} else {

console.log('---No match found at', i)

}

}

}

Tasks 1:

Just above the for loop in the letterFinder function definition, declare a variable named conditional and assign to it the following code:

**typeof(word) == 'string' && word.length >= 2**.

Task 2:

Declare a variable named condition2 on the next line and assign to it a check that makes sure that the type of match is a string AND that the length of the match variable is equal to 1.

Task 3:  
Write an if statement on the next line that checks that condition1 is true, and condition2 is true.

Task 4:

Move the rest of the functions body into the if statement you wrote in the previous step.

Task 5:

Code an “else” block after the “if” condition and console log the following: “Please pass correct arguments to the function.”

Task 6:

As a failing test, run the letterFinder function and pass it with any two numbers as arguments.

Task 7:

As a passing test, run the letterFinder function and pass it with correct arguments, such as: letterFinder(“cat”, “c”).

Completed:

function letterFinder(word, match) {

var condition1 = typeof(word) == 'string' && word.length >= 2;

var condition2 = typeof(match) == 'string' && match.length == 1;

if (condition1 == true && condition2 == true) {

for(var i = 0; i < word.length; i++) {

if(word[i] == match) {

//if the current character at position i in the word is equal to the match

console.log('Found the', match, 'at', i)

} else {

console.log('---No match found at', i)

}

}

} else {

console.log("Please pass correct arguments to the function.")

}

}

letterFinder([], []);

letterFinder("cat", "c");

Additional Resources

MDN Functions

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function>

MDN try…catch

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch>

Iteration protocols

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols>

The Math Object on MDN

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math>

The String Object on MDN

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String>

MDN JavaScript error reference

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Errors>

The null value on MDN

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null>

The undefined property on MDN

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined>