JavaScript

Console Log

// External files make your code organized and easier to maintain

console.log("This is the log for the 🔥EXTERNAL🔥JavaScript");

Hello Variable

// Declares student variable using var keyword

var studentName;

// Uses assignment operator(=) to assign a value

var studentName = "Abdul";

var studentAge = 32;

// To re-assign a variable, use only the variable's name

studentName = "Tonya";

studentAge = "forty-four";

// To access a value stored in a variable, use the variable's name

console.log(studentName);

//To combine the message with a variable value use the concatenation operator(+)

//Logs "My name is "

console.log("My name is ");

// Logs "My name is Tonya"

console.log("My name is " + studentName + " and I am " + studentAge);

Primitive Types

//Primitive data types include undefined, string, number and boolean

//Undefined variables haven't been assigned values yet.

var myUndefined;

// A string is a series of characters and is surrounded by quotes

var myStringWelcome = "Hello World";

var myStringPassword = "865Password!2020";

// A number can be either an integer or a decimal

var myNumberInt = 100;

var myNumberDecimal = 100.100;

// Booleans have two values: true or false

var isMyBooleanTrue = true;

var isMyBooleanFalse = false;

// To check the type of data, use typeof followed by the name of the variable

// Logs undefined

console.log(typeof myUndefined);

// Logs number

console.log(typeof myNumberInt);

// Logs boolean

console.log(typeof true);

// Logs string

console.log(typeof "Howdy");

// Pro-tip: JavaScript is loosely typed, so the type is tied to the value held in the variable, not the variable itself!

// Logs number

var myVariable = 33;

console.log(typeof myVariable);

// myVariable is reassigned; Logs boolean

myVariable = false;

console.log(typeof myVariable);

Logical Comparison Operators

var a = 100;

var b = 10;

var c = "10";

// Arithmetic operators combine with numbers to form an expression that returns a single number

console.log(a + b);

console.log(a - b);

console.log(a / b);

console.log(a \* b);

// Modulus returns the remainder between two numbers.

console.log(a % b);

console.log( 0 % 3); //0

// Comparison operators combine with strings, booleans and numbers to form an expression that evaluates to true or false

// Compares equality

// inside the parenthesis are expressions

console.log(b == c); // true  10 == "10"

console.log(b != c); // false

// Compares equality and type (strict equality)

console.log(b === c);  // false 10 === "10"

console.log(b !== c);  // true   10 is not not equal to "10"

// Greater than or less than

console.log(a > b);

console.log(a < b);

// Greater than or equal to and less than or equal to

console.log(a <= b);

console.log(a >= b);

// Logical operators take in two or more expressions and return true or false

var expression1 = (b == c); // true because it's two equal signs

var expression2 = (a > b);  // true

// Evaluates to true if expression1 AND expression2 are both true, otherwise false

console.log(expression1 && expression2);

// Evaluates to true if expression1 OR expression2 is true, otherwise false

console.log(expression1 || expression2);

// Logical Not (!) turns an expression that evaluates to true to false and vice versa

// Returns true

console.log(expression2);

// Returns false

console.log(!expression2);

Conditional Statements

var hungerLevel = 50;

var isLunchTime = true;

var lunchBill = 11;

// If statement

// Evaluates to true so "Hungry" is logged

if (hungerLevel >= 50) {

  console.log("Hungry!");

}

// Evaluates to false so nothing is logged

if (hungerLevel < 50) {

  console.log("Hungry!");

}

// Else statement

// Evaluates to true so "Lunchtime" is logged

if (isLunchTime === true) {

  console.log("Lunchtime");

} else {

  console.log("Not Lunchtime");

}

// isLunchTime is another way of writing "isLunchTime === true"

if (isLunchTime) {

  console.log("Lunchtime!!");

} else {

  console.log("Not Lunchtime!!");

}

// Evaluates to false so "It's Lunchtime Already" is logged

if (!isLunchTime) {

  console.log("Not Lunchtime Already!!");

} else {

  console.log("It's Lunchtime Already !!");

}

// Else if allows you to test more than one condition

// The first condition is false, so the second condition is evaluated. Logs "Cost Rating: $$""

if (lunchBill < 10) {

  console.log("Cost Rating: $");

} else if (lunchBill >= 10 && lunchBill < 15) {

  console.log("Cost Rating: $$");

} else {

  console.log("Cost Rating: $$$");

}

Arrays

// So far, we have been storing one piece of data in variables

var name = "Andre";

var pets = 3;

var isStudent = true;

// To store groups of data in a single variable, we use arrays

var names = ["Andre", "Karl", "Rashida", "Olivia"];

// The entire array can be accessed by using the array's name

console.log(names);

// To log a single element, we use the name of the array with the index in brackets

console.log(names[1]);

// Arrays are zero-indexed, so the index of first element in the array is 0

console.log(names[0]);

// We can also use index to replace data in an array

// Returns "Olivia"

console.log(names[3]);

//Replaces "Olivia" with "Carter"

names[3] = "Carter";

// Logs "Carter"

console.log(names[3]);

// We use the array's length property to determine how many elements are in the array

console.log(names.length);

Iteration

// We use a for-loop to execute code more than once

for (var i = 0; i < 5; i++) {

    // This is the block of code that will run each time

    console.log("This is the current value of i: " + i + ".");

}

// For-loops are often used to iterate over arrays

var zooAnimals = ["Bears", "Giraffes", "Penguins", "Meerkats"];

//To determine how many times the loop should execute, we use the array's length

// i must be less than the length in the zooanimals array or else you will get

// an error

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

    console.log("I am going to zoo to see " + zooAnimals[i] + ".");

}

//

for (var i = (zooAnimals.length - 1); i >= 0; i--) {

    console.log("I am going to the zoo to see " = names[i] + ".")

}

Functions

// Functions are reusable blocks of code that perform a specific task

// This is a function declaration

function declareHello() {

  console.log("Hello, I am a function declaration.");

  console.log("-----------------------------------");

  // Return stops the execution of a function

  return;

}

// This is a function expression

var expressHello = function() {

  console.log("Hello, I am a function expression.");

  console.log("-----------------------------------");

  return;

};

// Functions must be called to execute

declareHello();

expressHello();

//Functions can be called again to make the block of code execute again

declareHello();

// Functions can take parameters.

// Parameters give a name to the data to be passed into the function

function declareHelloAgain(x,y,z) {

  console.log("Hello, my parameter's values are " + x + ", " + y + ", and " + z);

  console.log("-----------------------------------");

  return;

}

// Function arguments give parameters their values

// Here the parameter x is given the value 7 when the function is called

declareHelloAgain(7, "Hello", true);

var favoriteTA = "David"

function addNumbers(a, b, c, d) {

  var sum = a + b + c + d;

  favoriteTA = "Goliath";

  console.log(favoriteTA);

  return sum;

}

console.log(addNumbers(75, 22.3, 33, 42));

Scope

// A variable declared in global scope is available to all functions

var hello = "Hello";

function sayHello() {

  console.log(hello);

  return;

}

var sayHelloAgain = function () {

  console.log(hello);

  return;

};

sayHello();

sayHelloAgain();

//A variable declared in local scope is only available to that function

function sayGoodbye() {

  var goodbye = "Goodbye";

  console.log(goodbye);

  return;

}

//This will throw an error

var sayGoodbyeAgain = function () {

  console.log(goodbye);

  return;

};

// Shadowing happens when the same variable is used in the local and global scope

var shadow = "Hello";

 console.log(shadow);

// Logs "Hello"

function sayWhat() {

  console.log(shadow);

  return;

}

//Logs "Goodbye"

var sayWhatAgain = function () {

  var shadow = "Goodbye";

  console.log(shadow);

};

sayGoodbye();

sayWhat();

sayWhatAgain();

Methods

var comparisonOperators = ["Equal", "Not Equal", "Strict Equal", "Strict Not Equal", "Greater Than", "Less Than"];

var arithmaticOperators = ["+", "-", "%"];

var logicalOperators = ["and", "or", "not"];

var myString = "Hello String";

//Array Methods

// Sorts comparisonOperators array and returns the sorted array

comparisonOperators.sort();

//Logs sorted array

console.log(comparisonOperators);

// Adds elements to end of an array. Takes in at least one parameter

arithmaticOperators.push("%");

//Logs array with element "%" added to end

console.log(arithmaticOperators);

//Returns selected elements as a new array.

var logicalOperatorsSliced = logicalOperators.slice(0,2);

//Logs new array

console.log(logicalOperatorsSliced);

// The orginal array is unchanged

console.log(logicalOperators);

//String Methods

//Replaces "String" with "World" and returns new string

var myNewString = myString.replace("String", "World");

console.log(myNewString);

console.log(myString[2]); // will pull the second element or character from the myString array above, it would be "l" in this case

//The orginal string is unchanged

console.log(myString);

Objects

// Objects are a collection of properties

// Object-literal syntax is a variable with a statement block, it looks just like a function statement.

var planet = {

  // Properties are made up of key-value pairs

  name: "Earth",

  age: "4.543 billion years",

  moons: 1,

  isPopulated: true,

  population: "7.594 billion"

  anotherObject: {

    ownProperties: "stuff",

  }

  myMethod: function() {

  }

};

// To access a property's value that is a string, number or booleean, use the object's name and the associated key

// Uses dot notation and logs "Earth"

console.log(planet.name);

// Uses bracket notation and logs "Earth"

console.log(planet["name"]);

Object Methods

// Objects can store more than primitive data types like strings, booleans and numbers

var planet = {

  name: "Earth",

  age: "4.543 billion years",

  moons: 1,

  isPopulated: true,

  population: "7.594 billion",

  // Objects can store arrays in a key-value pair

  neighboringPlanets: ["Mars", "Venus"],

  // Objects can also store methods

  tellFunFact: function () {

    console.log("The earth is the only planet in our solar system not named after a Roman god or goddess.");

  },

  showWarning: function () {

    console.log("Space junk falls into Earth's atmosphere at a rate of about one a day.");

  }

};

// To access a value in an array, use the name of the object, the key and the index.

// Logs "Mars" using dot notation

console.log(planet.neighboringPlanets[0]);

// Logs "Mars" using bracket notation

console.log(planet["neighboringPlanets"][0]);

// To call a method, use the name of the object and the key. Don't forget the ()!

planet.tellFunFact();

planet.showWarning();

Object This

// The default keyword "this" refers to the global object

// In a browser, the global object is the Window

// Logs Window

console.log(this);

// When the keyword "this" is used inside of an object like planet, "this" belongs to that object

var planet = {

  name: "Mars",

  age: "4.543 billion years",

  isPopulated: false,

  population: "7.594 billion",

  logFacts: function () {

     //Logs "This planet's name is Earth"

     console.log("This planet's name is " + this.name);

     //Logs "This planet's age is 4.543 billion years"

   console.log("This planet's age is " + this.age);

  },

  logPopulation: function () {

    if (this.isPopulated) {

      // Logs "This planet's population is 7.594 billion"

    console.log("This planet's population is " + this.population);

  } else {

    console.log("The planet is unpopulated");

  }

  }

};

// Calls object methods

planet.logFacts();

planet.logPopulation();