# Quoting Strings with Single Quotes (“ -> ‘)

var myStr = "<a href=\"http://www.example.com\" target=\"\_blank\">Link</a>";

* var myStr = '<a href="http://www.example.com" target="\_blank">Link</a>';

# Understand String Immutability

In JavaScript, String values are immutable, which means that they cannot be altered once created.

var myStr = "Bob";  
myStr[0] = "J";

*cannot change the value of myStr to "Job", because the contents of myStr cannot be altered.*

* var myStr = "Bob";  
  myStr = "Job";

# Use Bracket Notation to Find the Last Character in a String

// Example

var firstName = "Ada";

var lastLetterOfFirstName = firstName[firstName.length - 1];

# Nest one Array within Another Array

var ourArray = [["the universe", 42], ["everything", 101010]];

# Modify Array Data With Indexes

// Example

var ourArray = [1,2,3];

ourArray[1] = 3; // ourArray now equals [1,3,3].

# Access MultiDimensional Arrays With Indexes

var arr = [  
 [1,2,3],  
 [4,5,6],  
 [7,8,9],  
 [[10,11,12], 13, 14]  
];  
arr[3]; // equals [[10,11,12], 13, 14]  
arr[3][0]; // equals [10,11,12]  
arr[3][0][1]; // equals 11

# Manipulate Arrays With push

// Example

var ourArray = ["Stimpson", "J", "cat"];

ourArray.push(["happy", "joy"]);

// ourArray now equals ["Stimpson", "J", "cat", ["happy", "joy"]]

# Shift() – Pop()

That's where .shift() comes in. It works just like .pop(), except it removes the first element instead of the last.

# Unshift() – Push()

.unshift() works exactly like .push(), but instead of adding the element at the end of the array, unshift()adds the element at the beginning of the array.

// Example

var ourArray = ["Stimpson", "J", "cat"];

ourArray.shift(); // ourArray now equals ["J", "cat"]

ourArray.unshift("Happy");

// ourArray now equals ["Happy", "J", "cat"]

# Global Scope and Functions

// Declare your variable here

var myGlobal = 10;

function fun1() {

// Assign 5 to oopsGlobal Here

oopsGlobal = 5;

}

// Only change code above this line

function fun2() {

var output = "";

if (typeof myGlobal != "undefined") {

output += "myGlobal: " + myGlobal;

}

if (typeof oopsGlobal != "undefined") {

output += " oopsGlobal: " + oopsGlobal;

}

console.log(output);

}

# Local Scope and Functions

function myTest() {  
 var loc = "foo";  
 console.log(loc);  
}  
myTest(); // "foo"  
console.log(loc); // "undefined"

* loc is not defined outside of the function.

# Global vs Local Scope in Functions

It is possible to have both local and global variables with the same name. When you do this, the localvariable takes precedence over the global variable.

// Setup

var outerWear = "T-Shirt";

function myOutfit() {

// Only change code below this line

var outerWear = "sweater"; //biến khác cùng tên

outerWear = “sweater”; //thay giá trị “T-Shirt” => “sweater”

// Only change code above this line

return outerWear;

}

myOutfit();

# Return a Value from a Function with Return

# Stand in Line (queue)

function nextInLine(arr, item) {

// Your code here

arr.push(item);

var remove = arr.shift();

return remove; // Change this line

}

// Test Setup

var testArr = [1,2,3,4,5];

// Display Code

console.log("Before: " + JSON.stringify(testArr));

console.log(nextInLine(testArr, 6)); // Modify this line to test

console.log("After: " + JSON.stringify(testArr));

# Use Conditional Logic with If Statements

// Example

function ourTrueOrFalse(isItTrue) {

if (isItTrue) {

return "Yes, it's true";

}

return "No, it's false";

}

# Comparison with the Equality Operator ( == )

# Comparison with the Strict Equality Operator ( ====)

# Multiple Identical Options in Switch Statements

switch(val) {  
 case 1:  
 case 2:  
 case 3:  
 result = "1, 2, or 3";  
 break;  
 case 4:  
 result = "4 alone";  
}

# Returning Boolean Values from Functions

function isEqual(a,b) {  
 if (a === b) {  
 return true;  
 } else {  
 return false;  
 }  
}

* function isEqual(a,b) {  
   return a === b;  
  }

# Return Early Pattern for Functions

function myFun() {  
 console.log("Hello");  
 return "World";  
 console.log("byebye")  
}  
myFun();

The above outputs "Hello" to the console, returns "World", but "byebye" is never output, because the function exits at the return statement.

# Counting Cards

var count = 0;

function cc(card) {

// Only change code below this line

switch(card){

case 2:

case 3:

case 4:

case 5:

case 6:

count++;

break;

case 10:

case "J":

case "Q":

case "K":

case "A":

count--;

break;

}

if (count > 0){

return count + " Bet";

} else {

return count + " Hold";

}

// Only change code above this line

}

// Add/remove calls to test your function.

// Note: Only the last will display

cc(2); cc(3); cc(7); cc('K'); cc('A');

# Add New Properties to a JavaScript Object

// Setup

var myDog = {

"name": "Happy Coder",

"legs": 4,

"tails": 1,

"friends": ["Free Code Camp Campers"]

};

// Only change code below this line.

myDog.bark ="werr";

# Delete Properties from a JavaScript Object

// Setup

var myDog = {

"name": "Happy Coder",

"legs": 4,

"tails": 1,

"friends": ["Free Code Camp Campers"],

"bark": "woof"

};

// Only change code below this line.

delete ourDog.tails;

# Using Objects for Lookups

// Setup

function phoneticLookup(val) {

var result = "";

// Only change code below this line

var lookup = {

"alpha": "Adams",

"bravo": "Boston",

"charlie": "Chicago",

"delta": "Denver",

"echo": "Easy",

"foxtrot": "Frank"

};

result = lookup[val];

// Only change code above this line

return result;

}

// Change this value to test

phoneticLookup("charlie");

# Testing Objects for Properties

Sometimes it is useful to check if the property of a given object exists or not. We can use the .hasOwnProperty(propname) method of objects to determine if that object has the given property name. .hasOwnProperty() returns true or false if the property is found or not.

var myObj = {  
 top: "hat",  
 bottom: "pants"  
};  
myObj.hasOwnProperty(top); // true  
myObj.hasOwnProperty(middle); // false

# Accessing Nested Objects

var ourStorage = {  
 "desk": {  
 "drawer": "stapler"  
 },  
 "cabinet": {  
 "top drawer": {   
 "folder1": "a file",  
 "folder2": "secrets"  
 },  
 "bottom drawer": "soda"  
 }  
};  
ourStorage.cabinet["top drawer"].folder2; // "secrets"  
ourStorage.desk.drawer; // "stapler"

# Accessing Nested Arrays

var ourPets = [  
 {  
 animalType: "cat",  
 names: [  
 "Meowzer",  
 "Fluffy",  
 "Kit-Cat"  
 ]  
 },  
 {  
 animalType: "dog",  
 names: [  
 "Spot",  
 "Bowser",  
 "Frankie"  
 ]  
 }  
];  
ourPets[0].names[1]; // "Fluffy"  
ourPets[1].names[0]; // "Spot"

# Iterate with JavaScript For Loops

function multiplyAll(arr) {

var product = 1;

// Only change code below this line

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

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

console.log(arr[i][j]);

product += arr[i][j];

}

}

// Only change code above this line

return product;

}

multiplyAll([

[1, 21],

[3, 4],

[5, 6, 7]

]);

* 1
* 2
* 3
* 4
* 5
* 6
* 7

# Generate Random Fractions with JavaScript

Math.random();

Math.floor

*/\*\**

\* Returns the greatest number less than or equal to its numeric argument.

\* @param x A numeric expression.

*\*/*

floor(*x*: number): number;