JavaScript

**Variable**

**Definition**

“JavaScript variables are containers for storing data values. Variables are named containers that you can place data in and then refer to the data by naming the container.”

"A variable is a named value in your program. Whenever you use the name in the program, it's replaced with the value."

"You can think of a variable as a box that you can fit one thing in. If you put something else in it, the first thing goes away."

**Case Sensitive:**

* var myVariableis different than var mYVariable

**Invalid Variables:**

* Begin with a number
  + Bad:var 1dog;
  + Good:var dog1;
* Contain a JavaScript Keyword - A keyword is a reserved word in JavaScript. JavaScript uses these reserved words for its own purposes, and it would be confusing to you and the browser if you started using them for your variables.
  + eg.var function;
  + eg.var this;
  + full list at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical\_grammar#Reserved\_keywords\_as\_of\_ECMAScript?
* Contain a space between words
  + Bad: var my dog;
  + Good: var my\_dog;

**Safe Variable Beginings:**

* Begin with any letter.
* Begin with a dollar sign $.
* Begin with an underscore\_.

**Best Practice:**

* camelCase: First word is lowercase followed by capitalizing the first letter of each additional word.
* Bad practice: var lunchcostperstudent = 5;

All lowercase. Difficult to read.

* Bad practice: var LunchCostPerStudent = 5;

Should not capitalize first word.

* Best practice: var lunchCostPerStudent = 5;
* Human Readable: We code not for ourselves or the computer alone but for the eyes and understanding of others. "Try to give your variables name that describe what they're for or what they do. Other programmers may need to read and update your code in the future, and you'll thank yourself when you return to it at some point down the line. As your programs grow and involve more and more variables, objects, and functions, good variable names can really help you follow the flow of the program and understand its purpose. So keep those variable names simple, direct, and descriptive."
* Acceptable practice: var x = 25;

Though concise, this is not descriptive.

* Better practice: var age = 25;

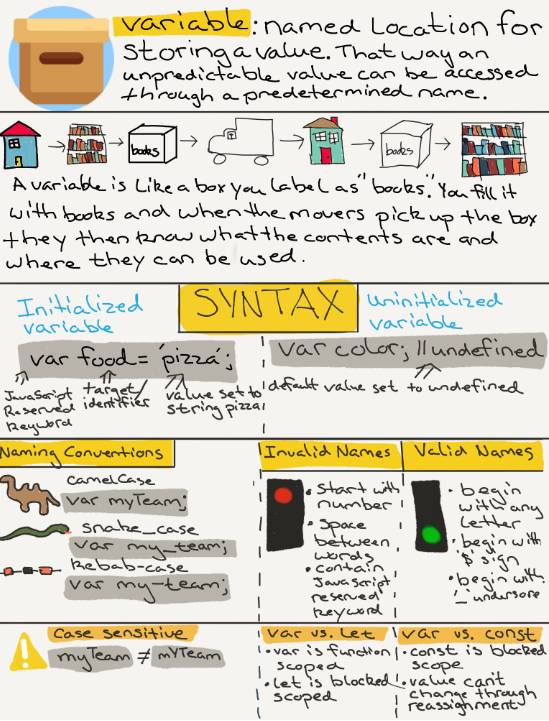
Not as concise, but makes code clearer.

**Summary:**

1. Variables let you store data for your program to use while it's running.

2. Declare a variable by following the var keyword with a name.

3. Choose simple, descriptive names for your variables, avoiding JavaScript's keywords and reserved words.



**String**

**Definition**

"Strings are a data type used to represent text and are wrapped in either a single or double quotes."

"A list of characters surrounded by quotes, used to represent text in computer programs."

"Strings in JavaScript (as in most programming languages) are just sequences of characters, which can include letters, numbers, punctuation, and spaces. We put strings between quotes so JavaScript knows where they start and end."

**Example**

* Single quote: var name ='Joe';
* Double quote: var name ="Joe";

**Best Practice**

* Choose one implementation, either single or double quote, and use consistently.
* Teams will often have agreed upon style guide.

**Error**

* Single and double quote: var name = 'Joe";
* Double and single quote: var name = "Joe';

**Turn Any Data Type Into a String**

1. typeof 24; // 'number'
2. typeof '24'; // 'string'
3. typeof true; // 'boolean'
4. typeof 'true'; // 'string'

**Single or Double Quotes Within Strings**

**Error**

* var greeting = 'It's good to meet you!';

Single quote within text wrapped in single quotes.

* var greeting2 = "Tell Jack that I say "Hi"";

Double quotes within text wrapped in double quotes.

**Solution**

* var greeting = "It's good to meet you!";

Single quote within double quotes.

* var greeting2 = 'Tell Jack that I say "Hi"';

Doubles quotes within single quotes.

**Escape Clause Solution**

* var greeting = 'It\'s good to meet you!';

Escape clause used before single quote within text wrapped in single quotes.

* var greeting2 = "Tell Jack that I say \"Hi\"";

Escape clause used before double quotes within text wrapped in double quotes.

**Looking Ahead - Future Lessons**

**Joining Strings**

1. var greeting = "Hi,";
2. var myName = "Rob";
3. greeting + " " + myName; // "Hi, Rob"

**Finding the Length of a String**

1. var neighborGreeting = "Hi, Rob";
2. neighborGreeting.length; // 7

**Getting a Single Character from a String**

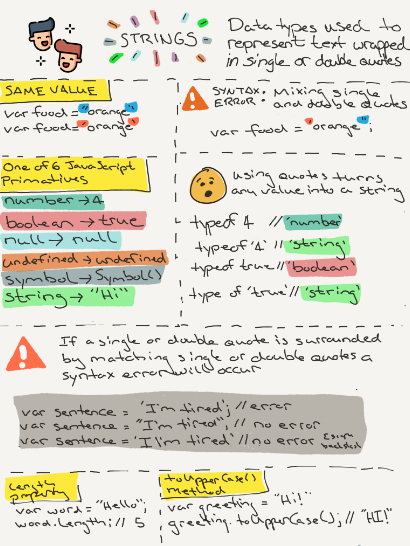
1. var neighborGreeting = "Hi, Rob";
2. neighborGreeting[0]; // "H"
3. neighborGreeting[1]; // "i"

**UpperCase**

1. var neighborGreeting = "Hi, Rob";
2. neighborGreeting.toUpperCase(); // "HI, ROB"

**Lower Case**

1. var neighborGreeting = "Hi, Rob";
2. neighborGreeting.toLowerCase(); // "hi, rob"



**Operators**

**Definition**

“Operators are used to assign values, compare values, perform arithmetic operations and more. Operators allow programmers to create a single value from one or more values.”

**Binary Operators:**

* Definition: Requires two operands, one before the operator and one after the operator.
* Syntax: operand1 operator operand2
* Example: 2 + 3; or x \* z;

**Unary Operators:**

* Definition: Requires a single operand, either before or after the operator.
* Syntax: operator operand OR operand operator
* Example: y++ OR ++y

**Arithmetic Operators: Multiplication, Division, Modulus, Addition and Subtraction**

* Definition: Takes numerical values (either literals or variables) as their operands and returns a single numerical value.
* Example: var count = 10 + 5 - 4; // 11
* Example: var mult = 3 \* 5; // 15

**Modulus Operators**

* Definition: This will give us the remainder of two values.
* Example: 21 % 4; // 1

**Assignment Operators:**

* Definition: Assigns a value to its left operand based on the value of its right operand.
* Example: var count = 5;

**Comparison Operators:**

* Definition: Compare both sides of equation and returns a logical value based on whether the comparison is true. The operands can be numerical, string, logical, or object values.
* Equality operator (==): Checks for equality in value. Coersion may take place finding equal values between a string and number.
* Strict equality operator (===): Checks for equality in value and type. Does not leave room for coersion.
* Example: 5 == '5'; // true
* example: 5 === '5' // false

**String Operator:**

* Definition: When the ‘+’ is used on strings the ‘+’ operator is called the concatenation operator.
* Example: 'Hello' + ' ' + 'world'; // 'Hello world'

**Object**

**Definition:**

"An object is a value type consisting of key/value pairs inside curly braces. The keys are also known as properties. Everything in JavaScript that isn’t a primitive is an object."

"A set of key-value pairs. Each key is a string that can be paired with any JavaScript value. You can then use the key to retrieve whatever value it's within in the object."

"Objects in JavaScript are very similar to arrays, but objects use strings instead of numbers to access the different elements. The strings are called keys or properties, and the elements they point to are called values. Together these pieces of information are called key-value pairs."

**Vocabulary Change:**

* Variables become known as properties in objects.
* Functions become known as methods in objects.

**Object Literal Notation:**

* Syntax: var object = {};
* Key/Properties: Single or double quotes not needed for valid variable names. Invalid variable names will require single or double quotes.
  + age = 24;
  + "user age" = 24;

**How to Access Properties on an Object:**

* The two primary ways of accessing properties of an object are with dot notation and bracket notation.
* Bracket Notation:
  + Syntax: object[‘property’] = value;
  + Example:

var book = { title: ‘Huck Fin’, pages: 260 };

book[‘title’]; // ‘Huck Fin’

book[’pages’]; // 260

* Dot Notation:
  + Syntax: object.property = value;
  + Example:

var name = { firtName: ‘John’, lastName: ‘Doe’ };

name.firstName; // ‘John’

name.lastName; // ‘Doe’

**How to Add Properties and Values on an Object:**

* You can add items to an object by using strings.
* Bracket Notation:
  + var person = {};
  + person["name"] = "Rob";
* Dot Notation:
  + var person = {};
  + person.name = "Rob";

**Object Constructor Definitivo**

A constructor is useful when you want to create multiple similar objects with the same properties and methods. The code creates objects as an instance of it.

**This keyword definition:**

Refers to an object. Which object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

In an object method, this refers to the object.

Alone, this refers to the global object.

In a function, this refers to theglobal object.

In a function, in strict mode, this is undefined.

In an event, this refers to the element that received the event.

Methods like call(), apply(), and bind() can refer this to any object.

**LOOPS**

Loops evaluate a condition. A true expression runs a code block. Loop repeat the procees until the expression is false.

**Condition**

Execute or skip statement depending on value of expression. These are decision points of your value code. Known as ‘paths’.

Boolean:

Truthy: everything else

Falsey: false, 0, “, null, undefined, NaN.

Logical Operators:

And (&&)

((5 > 4) && (8 > 3)) TRUE

((3 > 2) && (2 > 5)) FALSE

OR (||)

((5 > 4) || (8 > 3)) TRUE

((3 > 2) || (2 > 5)) TRUE

IF

IF ELSE

IF ELSE IF

**Switch Statements**

The switch statements is used to evaluate an expression the associating it with a case clause and finally running code executing statements matching that case.

Syntax

switch(expression) {  
 case x:  
 // code block  
 break;  
 case y:  
 // code block  
 break;  
 default:  
 // code block  
}

While/do while/for

while (condition) {

// Code block to execute

}

let text = "";  
let i = 0;  
do {  
 text += i + "<br>";  
 i++;  
} while(i < 5);

for(let i = 0; i < cars.length; i++) {  
 text += cars[i] + "<br>";  
}

**Regular Expressions**

Object that that describes a pattern of characters. Used to match a character cobination in strings. Performs matching and search and replace on text.

Syntax: “REGEX”

/*pattern*/moffiers;

var alphabet = /o*u/*i;

regular expression => /o*u/*i;

pattern => ou

modifier => i(case insensitive)

**Simple Pattern “/ou/”**

Simple pattern are constructed of characters for wich you want to find a direct match.

**This, bing, call, apply**

**Binding**: Asignar el valor que va a tomar THIS cuando se ejecute la función.

**Tipos de BINDING:**

1- Lexical Binding (Arrow Functions)

2- New Binding (Instanciar Objetos)

3- Explicit Binding (Invocación Indirecta)

4- Implicit Binding (Invocación de Método)

5- Default Binding (Invocación Directa)

**¿Cómo sabe qué enlazamiento aplicar?**

* Cómo fue escrita la función
* Modificaciones desde la creación
* El lugar de invocación (call site)

**5- Default Binding (Invocación Directa)**

Default binding refers to how this is the global context whenever a function is invoked without any of these other rules.

function quienSoy() {

console.log('Hola, yo soy', this);

}

quien soy();

Good practice is not use this in functions globals.

**4- Implicit Binding (Invocación de Método)**

Se produce cuando invocamos al método de un objeto.

const tito = {

nombre: "Victor",

saludar: function () {

console.log('Hola, mi nombre es ', ${ this.nombre });

}

};

tito.saludar();

Cuando trabajamos con eventos del DOM, this por defecto, es el elemento que dispara el evento.

**3- Explicit Binding (Invocación Indirecta)**

Sirve para que **nosotros elijamos** exáctamente qué objeto queremos que sea **this** cuando se ejecuta la función.

Podemos cambiar el contexto de forma explícita.

**Function.propertype.call:** Un **método** que nos permite invocar una función cambiándole el contexto.

object.saludar.**call**(object, parameter, parameter);

object.saludar.**apply**(object, [parameter, parameter]);

**Call y Apply invocan al método en el momento.**

**BIND NO invoca al método en el momento.**

Es un método de las funciones que nos retornan una nueva función con el contexto enlazado al objeto que le digamos.

object.saludar.**bind**(object, [parameter, parameter]);

Una función que fue creada con **bind** no puede volver a ser enlazada a otro objeto. Por ese motivo se lo conoce como el método de enlazamiento fuerte.

**2- New Binding (Instanciar Objetos)**

A new keyword is used to create an object from the constructor function.

let Cartoon = function(name, character) {

this.name = name;

this.character = character;

this.log = function() {

console.log(this.name + ' is a ' + this.character);

}

};

You can create objects using the new keyword like this:

let tom = new Cartoon('Tom', 'Cat');

let jerry = new Cartoon('Jerry', 'Mouse');

When a function is invoked with the new keyword, JavaScript creates an internal this object(like, this = {}) within the function. The newly created this binds to the object being created using the new keyword.

Sounds complex? Ok, let's break it down. Take this line,

let tom = new Cartoon('Tom', 'Cat');

Here the function Cartoon is invoked with the new keyword. So the internally created this will be bound to the new object being created here, which is tom.

**1- Lexical Binding (Arrow Functions)**

Se produce cuando escribimos una función como **arrow function.**

Arrow functions se ejecutan en el mismo contexto en las que fueron creadas.

Este enlazamiento también es fuerte.

No podemos cambiarle el contexto.

const tito = {

nombre: "Victor",

twitter: '@stanzianivm',

saludar: function () {

const seguimeEnTwitter = () => {

console.log('Hola, mi nombre es ', ${ this.nombre });

}

}

};

tito.saludar();

**THIS & BIND**

In most cases, the value of “THIS” is detemined by how a function is called. It can’t be set by assignment during execution, and it may be different each time the function is called. ES5 instroduced the bing method to set the value of a function’s “this” regardless of how it’s called. Bind creates a new function that will have “this” set to the first parameter passed to bind().

**ERRORES COMUNES**

const tito = {

nombre: "Victor",

twitter: '@stanzianivm',

saludar: function () {

const seguimeEnTwitter = () => {

console.log('Hola, mi nombre es ', ${ this.nombre });

}

}

};

tito.saludar();

**Functional Programming**

Javascript can handle higher-order functions, A higher-order is a function that can take another function as an argument, or that returns a function as a result.

**ERIC ELLIOT**

Functional programming glossaries contain a large number of large words, but at its core, the essence of FP is really very simple; programs are built mostly with a handfull of very small, very reusable, very predictable pure functions.

A pure function is a function in which:

* Given the same input, will always return the same output.
* Produce no side effects.
* Relies on no external state.

The filter()method creates a new array filled with elements that pass a test provided by a function.

The filter() method does not execute the function for empty elements.

The filter() method does not change the original array.

Map() creates a new array from calling a function for every array element.

Map() calls a function once for each element in an array.

Map() does not execute the function for empty elements.

Map() does not change the original array.

The reduce() method executes a reducer function for array element.

The reduce() method returns a single value: the function's accumulated result.

The reduce() method does not execute the function for empty array elements.

The reduce() method does not change the original array.

- List transformation

Applues a function against an accumulayot and eachvalue of array (from left to right) to reduce it to a single value.

**Algorithm**

An algorithm is a self-contained step-by-step set of operations to be performed. Algorithm perform calculation, data processing, and/or automated reasoning tasks.

In a computer an algorithm is a lot like a recipe and tells your computer precisely what steps to take to solve a problem or reach a goal.

**IndexOf:** The indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.

**LastIndexOf:** The lastIndexOf() method, given one argument: a substring to search for, searches the entire calling string, and returns the index of the last occurrence of the specified substring. Given a second argument: a number, the method returns the last occurrence of the specified substring at an index less than or equal to the specified number.

**Split method:** The split() method takes a pattern and divides a [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) into an ordered list of substrings by searching for the pattern, puts these substrings into an array, and returns the array.

**Sort method:** The sort() method sorts the elements of an array [*in place*](https://en.wikipedia.org/wiki/In-place_algorithm) and returns the reference to the same array, now sorted. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.

**Join method:** The join() method creates and returns a new string by concatenating all of the elements in an array (or an [array-like object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Indexed_collections" \l "working_with_array-like_objects)), separated by commas or a specified separator string. If the array has only one item, then that item will be returned without using the separator.