I-Novotek Academy

Full Stack Web Development Course PDF Guide

website: www.inovotekacademy.com

Youtube: i-novotek academy

Whatsapp: +233596038326/+8613051806737

Course link: https://www.udemy.com/course/fullstack-web-development-course-projects-base/?referralCode=F8C808368D020D5794BD

The Basics of Javascript: A Beginner's Guide

If you're new to programming, javascript is a great language to start with. It's easy to learn and there are many resources available to help you get started. In section, we'll cover the basics of javascript, including variables, data types, operators, and functions. By the end of this section, you'll have a better understanding of how javascript works and be able to write basic programs.

What is Javascript?

Javascript is a programming language that allows you to add dynamic content to your web pages. That means you can create things like animations, games, and form validation with javascript. It runs on your web browser and doesn't require a separate download or installation like some other languages.

`

Javascript is what's called a "client-side" language. That means the code runs on your computer, not on the server where the website is hosted. This is in contrast to "server-side" languages like PHP or ASP, which run on the server before the page is even sent to your browser.

Client-side languages are convenient because they don't require any special setup on the server. All you need is a text editor and a web browser, and you're ready to start coding!

How Does Javascript Work?

When you view a web page, your browser downloads the HTML code from the server and displays it as a webpage. Javascript code is embedded into the HTML code and runs automatically when the page loads.

The Benefits of Javascript

Javascript is a programming language that is widely used by web developers to create interactive web applications.

Javascript has a number of benefits that make it a popular choice for web development. It is easy to learn, versatile, and can be used to create a wide variety of applications. Additionally, javascript is free to use and is supported by all major web browsers.

If you are considering learning a programming language for web development, javascript is a great choice. It is a versatile language that can be used to create a wide variety of applications.

With Javascript you can become a frontend, backend and fullstack developer

With Javascript you can create web applications, mobile and desktop apps

Javascript Variables: A Comprehensive Guide

What are variables in JavaScript?

JavaScript variables are used to store data values. In JavaScript, data values can be text strings, numbers, arrays, or objects. Variables can be declared using the var, let, or const keywords.

How to create variables in JavaScript

We use the following keywords to create variables

- **var**
- **let**
- **const**

var keyword

the var keyword is used to create a variable that is globally scoped, meaning it can be accessed from anywhere in your code.

let keyword

The let keyword is used to create a variable that is locally scoped, meaning it can only be accessed from within the block of code it was declared in.

const keyword

The const keyword is used to create a variable that cannot be reassigned, making it a constant value.

To declare a variable, you will use the following syntax:

var variableName = value;

Where variableName is the name of your variable and value is the value you are assigning to the variable. You can also declare

multiple variables on the same line using a comma delimiter like this:

```
var name = "John Doe", age = 39, isMarried = true;
```

Once a variable is declared, you can assign it a value using the equal sign (=):

myName = "John Smith"; // Assigns the value "John Smith" to the myName variable.

You can also declare multiple variables without assigning values to them by leaving off the value part of the statement. Variables declared in this way will have a default value of undefined.

var name, age, isMarried; // all variables are assigned the value of undefined by default. /* The following shows data types */

Basic rules of JavaScript syntax

Here are some of the basic rules that govern JavaScript syntax.

1. JavaScript is case-sensitive. This means that language keywords, variables, function names, and any other identifiers must always be

typed with a consistent capitalization of letters. The keyword var is not the same as the keyword var.

2. Statements must end with a semicolon (;). For example, the statement x = 5; is valid, but the statement x = 5 is not.

3. Whitespace (spaces, newlines, and tabs) is generally ignored, except when it is used to separate tokens. For example, the statements x=5 and x = 5 are equivalent.

4. Comments can be added to your code to make it more readable. JavaScript supports two types of comments:

// This is an single-line comment /* This is a multi-line comment */

5. Identifiers can be any combination of letters, digits, underscores (_), and dollar signs ($). However, they cannot start with a digit. In addition, some reserved words cannot be used as identifiers (e.g., class, return, etc.).

6. Variables must be declared before they are used. This can be done using the keyword var. For

example:

var x; // declares a variable named x var y = 5; // declares a variable named y and assigns it the value 5

7. Data types in JavaScript include numbers, strings, Booleans (true/false values), and objects. There are also two special data types: undefined and null.

8. Numbers can be written with or without decimals. For example, 3.14, 42, and -99 are all valid numbers.

9. Strings must be enclosed in quotes. Single or double quotes can be used, but they must match (e.g., "hello" and 'goodbye' are both valid, but "hello' is not). Strings can span multiple lines by using the backslash (\ ) as an escape character:

"This is the first line.\nAnd this is the second." // produces "This is the first line.<newline>And this is the second."

10 .Code must be put within <script> tags in order to run

11 . Javascript is a text-based language. This means that it is made up of words, numbers, and punctuation marks that are read by a computer and interpreted into instructions.

Variable naming conventions in JavaScript

- **Variable names can start with letters**, $, or _ .

The following variable names are all valid:

```
$sname, sname, name_sname.
```

- **No spacing**

Variable names should not contain spaces use an underscore instead.

```
var user_name = "John Doe";
```

- **Use descriptive names:**

Using descriptive names for your variables is a good way to make your code more readable and understandable. When choosing a name for your variable, try to think of a word or phrase that accurately describes what the variable is used for.

- **Use camel case:**

Camel case is a naming convention where each word in the name is capitalized, except for the first word. For example, the variable name “myVariable” would be written in camel case.

Do not use reserved words:

When choosing a name for your variable, be sure to avoid using any of the reserved words in JavaScript. Reserved words are words that have special meaning in the language and cannot be used as variable names.

- **Do not use spaces or special characters:**

Spaces and special characters are not allowed in JavaScript variable names. If you need to use multiple words to describe your variable, you can use camel case orunderscores to separate the words (“my_variable”).

- **Keep it short:**

Long variable names can make your code more difficult to read and understand. When possible, try to choose a name that is short but still descriptive.

Javascript data types

The JavaScript language contains two different types of data: primitive values and objects.

A primitive value is a value that has no properties or methods. A primitive value is immutable, which means it cannot be changed. The only way to create a new primitive value is to create a new variable and assign it a new value.

There are the primitive values in JavaScript: undefined, null, Boolean, Number, String, Symbol, and bigint.

1. undefined

Undefined is a value that represents no value. It is the default value of variables that have not been assigned a value.

2. null

Null is a value that represents no value. It is used to indicate that a variable does not have a value.

3. Boolean

Boolean is a value that represents either true or false.

4. Number

A number is a value that represents a number. All numbers in JavaScript are 64-bit floating-point numbers.

5. String

String is a value that represents a sequence of characters. Strings are immutable, which means they cannot be changed. The only way to create a new string is to create a new variable and assign it a new value.

6. Symbol

Symbol is a unique and immutable value that can be used to identify an object. Symbols are typically used as keys in objects.

7. **Objects (collections of properties)**

Javascript Operators: A Comprehensive Guide

What are operators in JavaScript?

Operators in JavaScript are the symbols that represent certain actions that can be performed on variables. In JavaScript, there are many different kinds of operators, including arithmetic operators, assignment operators, comparison operators, logical operators, and Bitwise operators. Each kind of operator performs a different kind of operation.

Arithmetic operators

Arithmetic operators take two operands (values) and perform an arithmetic operation on them

Arithmetic operators include

- **Addition (+)** to add operands together example let sum = 2+6
- **Subtraction (-)** subtracts operands together let difference = 9-7
- Multiplication (*)
- **Division (/)** let total = 9/2
- **Modulus (%)** division with remainder. Example 12%2 would give you 0 as the remainder while 13%2 would give you 1 as it should be according to division;

Increment (++)

Synthax

- **x++ (** postfix**)** Postfix increment
- **++x (**prefix) Prefix increment

Postfix increment

If used postfix, the increment operator will first increment the value before returning it.

```
let x = 3;
```

y = x++;

```
// y = 3
// x = 4
```

Prefix increment

If used as a prefix operator (for example, ++x), the increment operator will increment the operand and return the new value.

```
let a = 2;
```

b = ++a;

```
// a = 3
// b = 3
```

Assignment operators

The assignment operator in javascript assigns a value to a variable. The most common assignment operator is the = operator, which assigns the value to the left of the = to the variable on the right. For example, if we have a variable called x and we want to give it the value of 5, we would write x = 5.

Other assignment operators include the += operator, which adds the value to the left of the = to the variable on the right, and the -= operator, which subtracts the value to the left of the = from the variable on the right. Example :

int x = 5; // Assigns 5 to x x -= 4; // Subtracts 4 from x, so now x equals 1 int y = 4; y += 2; // Adds 2 to y, so now y equals 6

Combined Assignment Operators

You can combine the assignment operator with basic arithmetic operators using the combined assignment operators (+= , -= , *= , /= , &= , and |=). This lets you write code like this:

let m=0

m = m + 5 is equivalent to m += 5 .

m = m * 3 is equivalent to m *= 3 .

Double equal to vs tripple equal to

Double equal to is == , it compares values.

Tripple euqal to is === , it compares both value and type.

Example of ==

"" == "0" is true then when comparing string , == will compare only values, not type.

```
 0 == false is also true, here as you can see when comparing
value 0 and false then it converts 0 to falsy value.
```

Example of ===

```
= "" === "0" is false, because it compares both values and
their data types(string vs number)
```

Comparison operators

Comparison operators Comparison operators are used to compare two values:

- greater than (>)
- less than (<)
- greater than or equal to (>=)
- and less than or equal to (<=).

NOTE: These operators evaluate expressions such that they return either true or false :

3 < 5 // evaluates to true 3 > 5 // evaluates to false 3 >= 4 // evaluates to true 2 <= 1 // evaluates

Logical operators

Logical operators allow JavaScript to perform boolean logic on values. These operators are:

- **and' ,bb'** : If each of the two operands is true, then the condition becomes true. (a && b) will be true only if both a and b are true.

- **or' ‡Ç¸Âf** any of the two operands is true, then the condition becomes true. (a || b) will be true if either a or b is true.

- **not' , "** Used with boolean values to reverse the value i.e. from false to true, and from true to false

  var x = 2;

```
// x != 7 is TRUE
```

Conditional Statements in javascript

What is a conditional statement? A conditional statement is a set of commands that will only be executed if a specified condition is met.

```
if (condition) {
  // commands to execute if condition is true
} else {
  // commands to execute if condition is false
}
```

If, else if, and else

are all used to create conditional statements in various programming languages.

Nested conditional statements A nested conditional statement is a conditional statement that contains another conditional statement as one of its components. Code example

Nested conditional statements

Truthy and falsy values in javascript

In JavaScript, a truthy value is a value that is considered true when evaluated in a Boolean context. All values are truthy unless they are defined as falsy. The following values are considered falsy:

- false

- 0

- ""

- null

- undefined

So what does this all mean? Basically, any value that is not one of the falsy values listed above is considered truthy. This includes values like true, 1, and "foo".

JavaScript Loops

JavaScript loops are a powerful programming tool that allows you to execute a block of code multiple times. While there are many different types of loops, they all share a common structure: a condition is checked, and if the condition is true, the code block is executed. This process is then repeated until the condition is no longer true. In this blog post, we'll take a closer look at how JavaScript loops work and how you can use them in your own pr

ograms.

The For Loop

The for loop has the following syntax: > The **for** statement creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement or a block of statements.

```
for ([initialization]; [condition]; [final-expression]){ //
code here... }
```

```
inal-expression ===incrementer/decrementer
```

> The **initialization** expression initializes the loop; it's executed once, as the loop begins.

> When the **condition** returns true, the loop executes the statement.

> The **final-expression** is executed every time the statement (s) runs.

Code example :

```
let i;
for(i=0;i<5; i++){ // code here will run 5 times! }
```

The while loop

The while loop is similar to the for loop, but instead of using an initializer and an incrementer, it only has a condition. The code block is executed repeatedly until the condition evaluates to false.

```
 while (condition) {
  code block to be executed}
  }
```

let i = 0;

```
  while (i < 10) {
    console.log(i);
    i++;
 } // loops ...0, 1, 2,...9
```

The While Loop

The while loop loops through a block of code as long as a specified condition is true.

```
while (condition) {
  // code  to run
}
```

```
let msg = "";
while (x < 10) {
 msg += "The number is " + i;
 i++;
}
```

Javascript Functions

What is a function?

In JavaScript, a function is a piece of code that is written to perform a specific task. Functions are usually self-contained and can be reused across your code. This makes them very important in JavaScript programming.

Why functions

There are many reasons why you should use functions in JavaScript. Here are 10 of the most important reasons:

1. Functions help to make your code more readable.

2. Functions can make your code more reusable.

3. Functions can help to make your code more maintainable.

4. Functions can improve the performance of your code.

5. Functions can help to modularize your code.

6. Functions can make your code more testable.

7. Functions can help to reduce the complexity of your code.

8. Functions can improve the flexibility of your code.

9. Functions can improve the

Defining functions

Functions can be written either as a

- function declaration or a
- function expression.

Function declarations are written as follows:

```
function name() {
  // code to be executed
}
```

Function expressions are written as follows:

```
let name = function() {
  // code to be executed
}
```

Function return keyword

The return keyword is used to exit a function and return a value to the caller.

The return keyword can be used with or without a value. If a value is present, it is returned to the caller. If no value is present, the function simply exits.

The return keyword is essential for creating functions that return values. Without it, functions would only be able to perform actions, not return values to the caller.

This would limit the usefulness of functions and make them much less powerful.

So if you're creating a function that needs to return a value, be sure to use the return keyword. It'll make your function much more useful and powerful.

Differences between function arguement and function parameters

Function arguments and function parameters are often used interchangeably, but there is actually a subtle difference between the two. Function arguments are the values that are passed to a function when it is invoked, while function parameters are the variables that are used to receive those arguments.

Javascript objects

What are objects?

Objects are variables that hold multiple pieces of information. In JavaScript, objects are created with curly braces {}.

Ways of creating object in javascript :-

1. Using an object literal:

```
let myObj = {}; // creates an empty object
```

2. Using the Object() constructor function:

```
let myObj = new Object(); // same as above
```

Creating objects using object literal:

In JavaScript, an object literal is a comma-separated list of name-value pairs enclosed in curly braces.

```
var objectName = {
key1: "value1",
key2: "value2"
};
```

Creating objects using object literal:

```
var person = {
name: "John",
age: 30,
city: "New York"
};
```

```
let myObj = new Object()
myObj.name='Joe'
```

Modifying objects

In this article, we will be discussing how to modify objects in JavaScript. Objects in JavaScript are mutable, which means they can be changed. There are a few ways to modify an object, and we will be covering some of them in this article.

We will be discussing the following topics:

- Adding and updating properties

- Deleting properties

By the end of this article, you should have a good understanding of how to modify objects in JavaScript. Adding and updating properties

There are a few ways to add and update properties in a JavaScript object. The most common way is to use the dot notation. With the dot notation, you can add or update a property by using the following syntax:

```
objectName.propertyName = value;

// Example
let obj = {};

obj.firstName = "John";

obj["lastName"] = "Doe";

console.log(obj); // {firstName: "John", lastName: "Doe"}
```

Alternatively, you can use the square bracket notation to add and update properties. The square bracket notation allows you to use variables for property names. This can be useful if you want to dynamically add or update properties based on user input. The syntax for using the square bracket notation is as follows:

```
objectName["propertyName"] = value;

//Example
let obj = {};

let firstName = "John";

let lastName = "Doe";
```

```
obj[firstName] = "First Name";
obj[lastName] = "Last Name";
console.log(obj); // {John: "First Name", Doe: "Last Name"}
```

Deleting properties There are a few ways to delete properties in JavaScript. The most common way is to use the delete keyword. You can delete a property by using the following syntax:

```
delete objectName["propertyName"];
// Example
let obj = {
  name: "John",
  dob: 1996,
  age: 24
};
console.log(obj.name); // 'John'
// We can now delete this `name` property;
delete object.name
 delete objectName["name"];
```

7. Loops and Conditionals

8. Functions

9. Objects

10. Arrays

11. Strings
12. Events
13. DOM Manipulation
14. CSS Manipulation
15. AJAX
16. Conclusion