**Documentation: JavaScript Fundamentals**

**Table of Contents:**

1. Variables
2. Data Types
3. Operators
4. Control Structures
5. Functions

**1. Variables**

Variables in JavaScript are containers for storing data values. They are declared using the **var**, **let**, or **const** keywords.

* **var:** Declares a variable globally or locally to an entire function regardless of block scope.
* **let:** Declares a block-scoped variable, limited to the block in which it is defined.
* **const:** Declares a constant variable whose value cannot be changed once it has been assigned.

Example:

var globalVar = 10;

let localVar = 20;

const PI = 3.14;

**2. Data Types**

JavaScript supports several data types, including:

* **Primitive Data Types:**
  + **Number:** Represents numeric values (e.g., integers, floats).
  + **String:** Represents text data enclosed in single or double quotes.
  + **Boolean:** Represents true or false values.
  + **Null:** Represents the intentional absence of any value.
  + **Undefined:** Represents a variable that has been declared but not assigned a value.
* **Composite Data Types:**
  + **Object:** Represents a collection of key-value pairs.
  + **Array:** Represents a collection of elements indexed by integers.
  + **Function:** Represents a reusable block of code.

Example:

let num = 10;

let str = "Hello, world!";

let bool = true;

let obj = { key: "value" };

let arr = [1, 2, 3];

**3. Operators**

JavaScript supports various types of operators:

* **Arithmetic Operators:** **+**, **-**, **\***, **/**, **%** (remainder)
* **Assignment Operators:** **=**, **+=**, **-=**, **\*=**, **/=**, **%=**
* **Comparison Operators:** **==**, **===**, **!=**, **!==**, **>**, **<**, **>=**, **<=**
* **Logical Operators:** **&&** (AND), **||** (OR), **!** (NOT)
* **Unary Operators:** **++**, **--**, **typeof**, **delete**
* **Ternary Operator:** **condition ? value1 : value2**

**1. Arithmetic Operators:**

Arithmetic operators perform mathematical operations on numeric operands.

* **Addition (+):** Adds two operands.
* **Subtraction (-):** Subtracts the right operand from the left operand.
* **Multiplication (\*):** Multiplies two operands.
* **Division (/):** Divides the left operand by the right operand.
* **Modulus (%):** Returns the remainder of the division of the left operand by the right operand.
* **Increment (++):** Increases the value of the operand by 1.
* **Decrement (--):** Decreases the value of the operand by 1.

**Example:**

let x = 10;

let y = 5;

let sum = x + y; // sum = 15

let difference = x - y; // difference = 5

let product = x \* y; // product = 50

let quotient = x / y; // quotient = 2

let remainder = x % y; // remainder = 0

x++; // Increment x by 1 (x = 11)

y--; // Decrement y by 1 (y = 4)

**2. Assignment Operators:**

Assignment operators assign values to variables.

* **Assignment (=):** Assigns the value of the right operand to the left operand.
* **Addition Assignment (+=):** Adds the value of the right operand to the left operand and assigns the result to the left operand.
* **Subtraction Assignment (-=):** Subtracts the value of the right operand from the left operand and assigns the result to the left operand.
* **Multiplication Assignment (\*=):** Multiplies the value of the left operand by the value of the right operand and assigns the result to the left operand.
* **Division Assignment (/=):** Divides the value of the left operand by the value of the right operand and assigns the result to the left operand.
* **Modulus Assignment (%=):** Calculates the modulus of the left operand with the right operand and assigns the result to the left operand.

**Example:**

let x = 10;

x += 5; // x = 15 (same as x = x + 5)

x -= 3; // x = 12 (same as x = x - 3)

x \*= 2; // x = 24 (same as x = x \* 2)

x /= 4; // x = 6 (same as x = x / 4)

x %= 2; // x = 0 (same as x = x % 2)

**3. Comparison Operators:**

Comparison operators compare two values and return a Boolean result (true or false).

* **Equal to (==):** Checks if two operands are equal.
* **Strict Equal to (===):** Checks if two operands are equal in value and type.
* **Not Equal to (!=):** Checks if two operands are not equal.
* **Strict Not Equal to (!==):** Checks if two operands are not equal in value or not of the same type.
* **Greater than (>):** Checks if the left operand is greater than the right operand.
* **Greater than or Equal to (>=):** Checks if the left operand is greater than or equal to the right operand.
* **Less than (<):** Checks if the left operand is less than the right operand.
* **Less than or Equal to (<=):** Checks if the left operand is less than or equal to the right operand.

**Example:**

let a = 10;

let b = 5;

console.log(a == b); // false

console.log(a === 10); // true

console.log(a != b); // true

console.log(a !== '10'); // true

console.log(a > b); // true

console.log(a <= b); // false

**4. Logical Operators:**

Logical operators perform logical operations on Boolean operands and return a Boolean result.

* **Logical AND (&&):** Returns true if both operands are true.
* **Logical OR (||):** Returns true if either of the operands is true.
* **Logical NOT (!):** Returns the opposite of the operand's Boolean value.

**Example:**

### let x = 10;

### let y = 5;

### console.log(x > 5 && y < 10); // true (Both conditions are true)

### console.log(x > 5 || y > 10); // true (At least one condition is true)

### console.log(!(x == y)); // true (x is not equal to y)

**5. Bitwise Operators:**

Bitwise operators perform bitwise operations on integer operands.

* **Bitwise AND (&):** Performs a bitwise AND operation.
* **Bitwise OR (|):** Performs a bitwise OR operation.
* **Bitwise XOR (^):** Performs a bitwise XOR (exclusive OR) operation.
* **Bitwise NOT (~):** Performs a bitwise NOT operation (bitwise inversion).
* **Left Shift (<<):** Shifts the bits of the left operand to the left by the number of positions specified by the right operand.
* **Right Shift (>>):** Shifts the bits of the left operand to the right by the number of positions specified by the right operand.
* **Zero-Fill Right Shift (>>>):** Shifts the bits of the left operand to the right by the number of positions specified by the right operand, filling the leftmost bits with zeros.

**Example:**

let a = 5; // Binary representation: 0101

let b = 3; // Binary representation: 0011

console.log(a & b); // 1 (Bitwise AND: 0101 & 0011 = 0001)

console.log(a | b); // 7 (Bitwise OR: 0101 | 0011 = 0111)

console.log(a ^ b); // 6 (Bitwise XOR: 0101 ^ 0011 = 0110)

console.log(~a); // -6 (Bitwise NOT: ~0101 = 1010)

console.log(a << 1); // 10 (Left Shift: 0101 << 1 = 1010)

console.log(a >> 1); // 2 (Right Shift: 0101 >> 1 = 0010)

console.log(a >>> 1); // 2 (Zero-Fill Right Shift: 0101 >>> 1 = 0010)

**Control Structures in JavaScript**

Control structures in JavaScript are constructs that enable you to control the flow of execution within your code. They allow you to make decisions, repeat blocks of code, and jump to different parts of your program based on conditions.

**1. Conditional Statements:**

Conditional statements allow you to execute different blocks of code based on specified conditions.

* **if Statement:** Executes a block of code if a specified condition is true.

if (condition) {

// code block to be executed if condition is true

}

**if...else Statement:** Executes one block of code if a specified condition is true and another block of code if the condition is false.

if (condition) {

// code block to be executed if condition is true

} else {

// code block to be executed if condition is false

}

**else if Statement:** Allows you to specify multiple conditions, with each condition being checked if the previous ones were false.

if (condition1) {

// code block to be executed if condition1 is true

} else if (condition2) {

// code block to be executed if condition2 is true

} else {

// code block to be executed if all conditions are false

}

**switch Statement:** Evaluates an expression and executes a block of code associated with the matched case.

switch (expression) {

case value1:

// code block to be executed if expression equals value1

break;

case value2:

// code block to be executed if expression equals value2

break;

default:

// code block to be executed if expression doesn't match any case

}

**2. Loops:**

Loops allow you to repeat a block of code multiple times until a specified condition is met.

* **for Loop:** Executes a block of code a specified number of times.

for (initialization; condition; increment/decrement) {

// code block to execute

}

**while Loop:** Executes a block of code as long as a specified condition is true.

while (condition) {

// code block to execute

}

**do...while Loop:** Executes a block of code once, and then repeats the loop as long as a specified condition is true.

do {

// code block to execute

} while (condition);

**for...in Loop:** Iterates over the enumerable properties of an object.

for (variable in object) {

// code block to execute

}

**for...of Loop (ES6):** Iterates over the iterable objects (arrays, strings, etc.)

for (variable of iterable) {

// code block to execute

} **3. Jump Statements:**

Jump statements allow you to transfer control to another part of your code.

* **break Statement:** Terminates the current loop, switch, or label statement, and transfers control to the statement immediately following the terminated statement.
* **continue Statement:** Skips the current iteration of a loop and continues with the next iteration.
* **return Statement:** Exits the current function and returns a value to the caller.

Control structures are fundamental to programming and are used extensively in JavaScript and other programming languages to create complex algorithms and control the flow of execution within a program.

Top of Form

**Functions**

**Table of Contents:**

1. Introduction to Functions
2. Function Declaration
3. Function Expression
4. Arrow Functions
5. Function Parameters
6. Return Statement
7. Function Scope
8. Function Hoisting
9. Callback Functions
10. IIFE (Immediately Invoked Function Expression)
11. Named vs. Anonymous Functions

**1. Introduction to Functions:**

Functions in JavaScript are reusable blocks of code that perform a specific task or return a value. They help in organizing code, promoting reusability, and making programs more modular.

**2. Function Declaration:**

Function declaration defines a function with a specified name, parameters (if any), and a function body.

**3. Function Expression:**

Function expression defines a function as part of an expression. It can be assigned to a variable.

**4. Arrow Functions:**

Arrow functions provide a more concise syntax for defining functions. They are especially useful for inline functions and functions with implicit return values

**5. Function Parameters:**

Functions can accept zero or more parameters, which act as placeholders for values passed to the function when it is called.

**6. Return Statement:**

Functions can return a value using the **return** statement. The return statement ends the function's execution and specifies the value to be returned to the caller.

**7. Function Scope:**

Variables declared inside a function are scoped to that function and are not accessible outside of it, unless explicitly returned or accessed through closures

**8. Function Hoisting:**

In JavaScript, function declarations are hoisted to the top of their scope, meaning they can be called before they are declared in the code.

**9. Callback Functions:**

A callback function is a function passed as an argument to another function, which is then invoked inside the outer function to complete some kind of routine or action.

**10. IIFE (Immediately Invoked Function Expression):**

An IIFE is a function that is immediately executed after it's created. It helps to create a private scope and avoid polluting the global namespace.