Lesson 1- JavaScript Control Structures

---

1. Introduction to Control Structures:

- **Definition:** Control structures in JavaScript determine the flow of code execution based on conditions or repetitions.
- **Importance:** They enable dynamic decision-making in code, allowing for different outcomes based on user input, data, or other conditions.

---

2. Conditional Statements:

- **Definition:** These statements allow the execution of specific blocks of code based on whether a condition is true or false.
- **If/Else Statements:**

Syntax:
```
if (condition) {
  // code for true condition
} else {
  // code for false condition
}
```

Usage: Ideal for binary decisions.

Extended Syntax (Else If): Used for multiple conditions.
```
if (condition1) {
  // code for condition1
} else if (condition2) {
  // code for condition2
} else {
  // code if none of the conditions are true
}
```

---

3. Ternary Expressions:

- **Definition:** A concise way to represent an if/else statement, often used for simple conditions.

Syntax:
```
condition ? expressionIfTrue : expressionIfFalse
```

Usage: Best for situations where you want to assign a value based on a condition.

Example:
```
let isAdult = (age >= 18) ? true : false;
```

---

4. Switch Statements:

- **Definition:** A control structure used when comparing a single value against multiple possible outcomes.

Syntax:

```
switch(expression) {
  case value1:
    // code for value1
    break;
  case value2:
    // code for value2
    break;
  default:
    // code if no match
}
```

Usage: Ideal when checking a variable or expression against multiple specific values.

Break Keyword: Essential after each case to prevent “falling through” to subsequent cases.

Default Case: Executes if no other cases match. It’s a safety mechanism.

---

5. Best Practices:

- **Code Clarity:** Always prioritize readability over brevity.
- **Comments:** Use them to explain the purpose of a block of code, especially if it might be complex or unclear to others.
- **Consistent Syntax:** Stick to a consistent code style for better maintainability.

Lesson 2- JavaScript Control Structures - Loops

---

1. Introduction to Loops:

- **Definition:** Loops are control structures that execute blocks of code multiple times based on specified conditions.
- **Importance:** They automate repetitive tasks, making code more efficient and concise.
- **Types of Loops in JavaScript:** for, while, and do/while loops.

---

2. For Loops:

- **Definition:** Executes a block of code a predetermined number of times.

Syntax:
```
for (initialization; condition; iteration) {
  // code to be executed
}
```

Components:

- **Initialization:** Establishes the loop control variable.
- **Condition:** Determines if the loop should continue.
- **Iteration:** Updates the control variable after each loop cycle.

Usage: Ideal when the number of iterations is known.

Example: Counting from 0 to 9.
```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

Iterating Over Arrays:
```
let numbers = [1, 2, 3, 4, 5];
let sum = 0;
for (let i = 0; i < numbers.length; i++) {
  sum += numbers[i];
}
console.log(sum);
```

Control Statements:

- **Break:** Exits the loop prematurely.
- **Continue:** Skips the current iteration and proceeds to the next.

---

3. While Loops:

- **Definition:** Executes a block of code as long as a specified condition is true.

Syntax:
```
while (condition) {
  // code to be executed
}
```

Usage: Ideal when the number of iterations is unknown but a condition is available.

Example: Counting from 0 to 9.
```
let i = 0;
while (i < 10) {
  console.log(i);
  i++;
}
```

Iterating Over Arrays:
```
let numbers = [1, 2, 3, 4, 5];
let sum = 0;
let i = 0;
while (i < numbers.length) {
  sum += numbers[i];
  i++;
}
console.log(sum);
```

---

4. Do/While Loops:

- **Definition:** Similar to while loops but guarantees the block of code is executed at least once.

Syntax:
```
do {
  // code to be executed
} while (condition);
```

Usage: Ideal when the code block should run at least once regardless of the condition.

Example: Prompting user input.
```
let number;
do {
  number = prompt("Enter a number:");
} while (isNaN(number));
console.log("You entered the number: " + number);
```

Iterating Over Arrays:
```
let numbers = [1, 2, 3, 4, 5];
let sum = 0;
let i = 0;
do {
  sum += numbers[i];
  i++;
} while (i < numbers.length);
console.log("The sum of the numbers is: " + sum);
```

---

5. Conclusion & Key Takeaways:

- **Power of Loops:** They allow us to write efficient, concise, and DRY (Don't Repeat Yourself) code.
- **Choosing the Right Loop:**
  - Use **for loops** when the number of iterations is known.
  - Use **while loops** when only the end condition is known.
  - Use **do/while loops** when the block must execute at least once.

Lesson 3 - Functions in JavaScript

---

1. Introduction to Functions:

- **Definition**: Functions are modular blocks of code designed to perform specific tasks.
- **Importance**:
  - Improve code readability.
  - Enhance code maintainability.
  - Avoid code repetition.
- **Goal**: By the end of this lesson, learners should be proficient in creating and invoking functions, understanding scope, parameters, arguments, and various function syntaxes.

---

2. Creating Functions:

There are several methodologies to craft functions in JavaScript:

Function Declaration:

Syntax:
```
function functionName() {
  // code block
}
```

Invocation: To execute, call the function using its name followed by parentheses (`functionName();`).

Example:
```
function greet() {
  console.log("Hello world!");
}
```

Function Expression:

Functions can be assigned to variables.

Syntax:
```
const variableName = function() {
  // code block
};
```

Example:
```
const greet2 = function() {
  console.log("Hello world!");
};
```

Arrow Functions (ES6):

A concise way to create functions.

Doesn’t have its own `this` context.

Can’t be used with the `new` keyword.

Doesn’t have an `arguments` object; use rest parameters instead.

Syntax:
```
const variableName = () => {
  // code block
};
```

Example:
```
const greet3 = () => {
  console.log("Hello world!");
};
```

Immediately Invoked Function Expressions (IIFE):

Functions that execute immediately upon definition.

Useful for creating private scopes.

Syntax:
```
(function() {
  // code block
})();
```

Example:
```
(function() {
  console.log("I run immediately!");
})();
```

---

3. Hoisting:

- **Definition**: In JavaScript, function declarations are hoisted, allowing them to be invoked before their actual definition.
- **Limitations**: Function expressions and arrow functions are not hoisted. They must be defined before invocation.

---

4. Scope:

- **Definition**: Refers to the visibility and accessibility of variables.
- **Types**:
  - **Local Scope**: Variables defined inside a function; not accessible outside.
  - **Global Scope**: Variables defined outside a function; accessible globally.

---

5. Parameters and Arguments:

- **Parameters**: Placeholders in the function definition.
- **Arguments**: Actual values passed when the function is invoked.
- **Default Parameters**: Set default values for parameters if no arguments are provided during invocation.
- **The `return` Keyword**: Allows functions to return values to the caller.

---

6. Conclusion & Takeaway:

- Functions are essential in programming, promoting code organization, reusability, and readability.
- Understanding function types, scope, and parameter handling is vital for effective JavaScript programming.
- Upcoming lessons will delve into how functions interact with other data structures, such as arrays and objects.

Resources

Module 9 CodePen Link

Free Code Camp:
JavaScript Algorithms and Data Structures Certification | freeCodeCamp.org

JavaScript Tutorial (W3):
JavaScript Tutorial