Control Flow in JavaScript

Control flow is the order in which the computer executes statements in a script. By default, JavaScript executes code from top to bottom. However, control structures allow us to alter this flow to make decisions, repeat tasks, or handle exceptional conditions.

---

1. Conditional Statements

Conditional statements allow code to make decisions based on specific conditions.

1.1 `if` Statement

The `if` statement executes a block of code if the condition is true.

Syntax:

```
if (condition) {
    // Code to execute if the condition is true
}
```

Example:

```
let age = 20;

if (age >= 18) {
    console.log("You are an adult.");
}
```

Flow:

- The condition is evaluated.
- If `true`, the block inside `{}` is executed; otherwise, it is skipped.

1.2 `if...else` Statement

The `else` block executes if the condition is false.

Syntax:

```
if (condition) {
    // Code if true
} else {
    // Code if false
}
```

Example:

```
let age = 16;

if (age >= 18) {
    console.log("You can vote.");
} else {
    console.log("You are too young to vote.");
}
```

Flow:

- Evaluate the condition.
- If `true`, execute the `if` block.
- If `false`, execute the `else` block.

1.3 `if...else if...else` Statement

This handles multiple conditions.

Syntax:

```
if (condition1) {
    // Code for condition1
} else if (condition2) {
    // Code for condition2
} else {
    // Code if none of the above are true
}
```

Example:

```
let score = 85;

if (score >= 90) {
    console.log("Grade: A");
} else if (score >= 75) {
    console.log("Grade: B");
} else {
    console.log("Grade: C");
}
```

Flow:

- Evaluate `condition1`. If true, execute its block.

- If false, move to `condition2`, and so on.
- If no conditions match, execute the `else` block.

1.4 Ternary Operator

A shorthand for `if...else`.

Syntax:

```
condition ? expressionIfTrue : expressionIfFalse;
```

Example:

```
let age = 20;
let access = (age >= 18) ? "Allowed" : "Denied";
console.log(access);
```

---

2. Switch Statement

The `switch` statement is used to evaluate multiple cases.

Syntax:

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

Example:

```
let day = 3;

switch (day) {
    case 1:
        console.log("Monday");
        break;
    case 2:
        console.log("Tuesday");
```

```
        break;
    case 3:
        console.log("Wednesday");
        break;
    default:
        console.log("Invalid day");
}
```

Flow:

- Evaluate `expression`.
- Compare with `case` values.
- Execute the matching block until `break` is encountered or continue to `default` if no match.

---

3. Loops

Loops are used for repeating tasks.

3.1 `for` Loop

Used for a known number of iterations.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // Code to execute
}
```

Example:

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

Flow:

1. Initialize `i = 0`.
2. Check the `condition` (`i < 5`).
3. Execute the loop body.
4. Increment `i` and repeat until `condition` is false.

3.2 `while` Loop

Repeats while the condition is true.

Syntax:

```
while (condition) {
    // Code to execute
}
```

Example:

```
let i = 0;

while (i < 5) {
    console.log(i);
    i++;
}
```

Flow:

1. Evaluate the `condition`.
2. If `true`, execute the block.
3. Repeat until `condition` is false.

3.3 `do...while` Loop

Similar to `while`, but executes at least once.

Syntax:

```
do {
    // Code to execute
} while (condition);
```

Example:

```
let i = 0;

do {
    console.log(i);
    i++;
} while (i < 5);
```

Flow:

1. Execute the block.

2. Check the `condition`.
3. If true, repeat.

3.4 `for...of` Loop

Iterates over iterable objects like arrays.

Syntax:

```
for (variable of iterable) {
    // Code to execute
}
```

Example:

```
let fruits = ["Apple", "Banana", "Cherry"];

for (let fruit of fruits) {
    console.log(fruit);
}
```

3.5 `for...in` Loop

Iterates over object properties.

Syntax:

```
for (key in object) {
    // Code to execute
}
```

Example:

```
let person = { name: "John", age: 25 };

for (let key in person) {
    console.log(`${key}: ${person[key]}`);
}
```

---

4. `break` and `continue`

4.1 `break`

Exits a loop or `switch`.

Example:

```
for (let i = 0; i < 5; i++) {
    if (i === 3) break;
    console.log(i);
}
```

4.2 `continue`

Skips the current iteration.

Example:

```
for (let i = 0; i < 5; i++) {
    if (i === 3) continue;
    console.log(i);
}
```

---

5. Exception Handling

Handles errors gracefully.

5.1 `try...catch`

Used to catch and handle errors.

Syntax:

```
try {
    // Code to try
} catch (error) {
    // Code to handle errors
} finally {
    // Code that always executes
}
```

Example:

```
try {
    let result = 10 / 0;
```

```
    console.log(result);
} catch (error) {
    console.log("Error occurred:", error.message);
} finally {
    console.log("Execution complete.");
}
```

Summary Flowchart

1. **Conditionals:**
    - `if` → `else` → `else if` → `switch`.
2. **Loops:**
    - `for` → `while` → `do...while` → `for...of` → `for...in`.
3. **Error Handling:**
    - `try...catch`.

With these tools, you can structure any logical flow in your JavaScript programs. Use them to build interactive and efficient scripts!