PDF 07

JavaScript Control Flow

Making Decisions with if, else, else if & switch

Learn how JavaScript makes decisions, controls program execution, and chooses different paths based on conditions.

Table of Contents

01 Introduction

Imagine opening your favorite food delivery app.

The application doesn't show every screen to every user. Instead, it makes decisions.

If you're already logged in, it takes you directly to the home page. If you're a new user, it asks you to create an account. If your internet connection is unavailable, it displays an error message. If a restaurant is closed, it prevents you from placing an order.

The application is constantly deciding what should happen next. This ability to choose different paths based on conditions is called Control Flow.

Until now, you've learned how JavaScript stores data using variables and data types, performs calculations with operators, and compares values using comparison and logical operators.

But knowing how to compare values isn't enough. Your program also needs to know what action to take after those comparisons.

For example:

— If the password is correct, log the user in.

— If the score is greater than 80%, award a certificate.

— If the shopping cart is empty, disable the checkout button.

— If the user isn't logged in, redirect them to the login page.

These aren't calculations—they're decisions. Control Flow is the mechanism that allows JavaScript to make those decisions.

Without it, every program would simply execute every line of code from top to bottom, whether it made sense or not.

Learning Control Flow is one of the biggest milestones in becoming a programmer because it's the first time your code begins to react to different situations instead of always behaving the same way.

02 What Is Control Flow?

Every JavaScript program consists of instructions.

When you run the program, JavaScript starts at the top and normally executes each instruction one after another. This default behavior is called sequential execution.

For example, imagine a recipe:

1. Wash the vegetables.
2. Cut the vegetables.
3. Heat the pan.
4. Cook the vegetables.
5. Serve the food.

Each step happens in order. You don't jump from Step 1 to Step 5. Programs work the same way.

However, real applications cannot always follow a fixed sequence. Sometimes they need to ask questions before deciding what to do next.

For example:

— Is the user logged in?

— Is the payment successful?

— Is the form complete?

— Does the requested product exist?

Depending on the answer, the application may follow a completely different path. This ability to change the execution path is called Control Flow.

Think of Control Flow as the traffic system of your program. Just as traffic lights, signs, and intersections determine where vehicles should go, Control Flow determines which parts of your code should execute and which parts should be skipped.

Without Control Flow, every application would behave the same way for every user, regardless of their actions or input.

03 Why Control Flow Matters

Imagine building haas.dev without Control Flow.

Every visitor would immediately receive:

— Every course

— Every certificate

— Every quiz result

— Every dashboard

— Every admin feature

Clearly, that wouldn't make sense. Instead, the application needs rules.

For example:

— If the learner is logged in, show the dashboard.

— Otherwise, show the login page.

Or:

— If the learner completes the course, unlock the certificate.

— Otherwise, continue showing the remaining lessons.

Notice something important. The application isn't simply displaying information. It's making decisions based on conditions. This is exactly what Control Flow enables.

It transforms a static program into an interactive application that responds differently to different users and situations.

04 How Programs Execute Code

Before learning if statements, it's important to understand how JavaScript executes instructions. By default, JavaScript follows a simple rule:

> Execute one line at a time, from top to bottom.

Imagine reading a book. You don't normally jump from page 5 to page 80. You read each page in order. JavaScript behaves similarly.

However, Control Flow introduces new possibilities. Instead of always moving straight downward, the program can:

- Skip certain instructions.
- Execute different instructions.
- Repeat instructions.
- Stop execution early.

These behaviors make software dynamic instead of static.

05 Sequential Execution vs Decision Making

Think about entering a shopping mall.

— Some doors are open to everyone.

— Other doors require an employee ID.

— Some areas are restricted to managers.

Although everyone enters through the same building, not everyone follows the same path. Applications work exactly the same way.

Every user starts the program. But depending on their permissions, actions, and data, they may experience completely different flows.

This flexibility is what makes modern software possible.

06 The if Statement

The if statement is JavaScript's most basic decision-making tool. It runs a block of code only when a condition is true.

```
if (condition) {
  // code runs if condition is true
} else if (anotherCondition) {
  // code runs if the first condition is false
  // and this one is true
} else {
  // code runs if none of the above are true
}
```

JavaScript checks each condition in order, from top to bottom. The moment one condition evaluates to true, that block runs — and the rest are skipped entirely.

The switch statement offers an alternative when you're checking one value against many possible matches, which can be cleaner than a long chain of else if statements.

07 Real-World Example: Login System

Let's connect this to something you interact with every day: logging into an app.

```
if (!username || !password) {
  showError('Please fill in all fields');
} else if (password !== correctPassword) {
  showError('Incorrect password');
} else {
  redirectTo('/dashboard');
}
```

Notice how each condition is checked in sequence. The system first makes sure fields aren't empty, then verifies the password, and only then grants access.

This is Control Flow doing exactly what it's meant to do — guiding the program down the correct path based on real conditions.

08 Common Beginner Mistakes

× **Using = instead of ==/===**

A single equals sign assigns a value; it doesn't compare one. This silently changes your variable instead of checking it.

× **Forgetting curly braces on multi-line blocks**

Without braces, only the very next line belongs to the if statement — everything after runs regardless.

× **Writing overly nested if statements**

Deeply nested conditions become hard to read and debug. Look for ways to simplify or restructure the logic.

× **Not considering the else case**

Beginners often handle the 'true' path and forget what should happen when the condition is false.

09 Summary

- Control Flow determines which parts of your code run, based on conditions.
- By default, JavaScript executes code sequentially, from top to bottom.
- if, else if, and else let a program branch down different paths.
- switch is a clean alternative for matching one value against many options.
- Real applications rely on Control Flow constantly — logins, permissions, forms, and more.

10 Cheat Sheet

```
if (condition) { ... }
if (condition) { ... } else { ... }
if (a) { ... } else if (b) { ... } else { ... }
switch (value) {
  case 'x': ...; break;
  default: ...;
}
```

— Conditions are evaluated top to bottom — the first true block wins.

— Always use === for comparisons, not =.

— Wrap multi-line blocks in { } every time.

— Use switch when comparing one variable to many fixed values.

Checklist

- [ ] I understand what Control Flow means and why it matters.
- [ ] I can explain sequential execution vs. decision-making.
- [ ] I can write an if / else if / else chain confidently.
- [ ] I know when to reach for switch instead of else if.
- [ ] I understand common mistakes like using = instead of ===.
- [ ] I can trace through a real example (like a login system) step by step.

Mini Architecture Challenge

You're designing the access logic for haas.dev's course dashboard. Think through the conditions before writing any code.

The scenario:

— A visitor who isn't logged in should be redirected to the login page.

— A logged-in user who hasn't paid should see a locked preview.

— A logged-in, paid user should see the full dashboard.

— A paid user who has finished every lesson should also see a 'Get Certificate' button.

Your task:

— Sketch the order in which these conditions should be checked.

— Decide: does this need if/else if/else, or would nested ifs work better here?

— Write the pseudocode before touching real JavaScript.

There's no single right answer — the goal is to practice thinking in conditions before writing syntax.

11 Related Resources

Before continuing, review these related haas.dev PDFs if needed:

- → JavaScript Comparison, Logical & Modern Operators

  Learn how conditions are created before they're used for decision-making.

- → JavaScript Data Types: Understanding Every Kind of Data Your Program Can Store

  Understand how different values behave inside conditions.

- → Variables & Memory: How JavaScript Stores, Remembers, and Uses Data

  Learn where the values used in decisions come from.

Recommended Learning Path

PDF 05	JavaScript Data Types
PDF 06	Comparison, Logical & Modern Operators
PDF 07	JavaScript Control Flow ← you are here
PDF 08	Loops: for, while & Iteration
PDF 09	Functions: Reusable Logic Blocks

haas.dev

Understand → Break → Design → Build