Conditionals — if/else, Operators, Ternary & switch

AiCE Software Engineering Foundations — JavaScript & Node.js

Arrays and objects are covered — you even saw an object hold methods that operate on its own data. This lesson looks at control flow: how a JavaScript program makes decisions. The logic here is exactly what you already know from Python. Only the syntax is different.

1. if, else if, else

Create a file named conditionals.js:

```
const score = 75;

if (score >= 70) {
    console.log('Grade: A');
} else if (score >= 60) {
    console.log('Grade: B');
} else {
    console.log('Grade: F');
}
```

Save and run — this prints 'Grade: A', since 75 is greater than or equal to 70.

Difference from Python	What Changes
The condition	Goes inside parentheses: if (score >= 70) instead of if score >= 70:.
The block of code	Goes inside curly brackets {}, instead of being marked out by indentation.
Python's elif	Becomes else if, written as two separate words.

Apart from these three differences, the logic is identical to what you already know.

2. Comparison & Logical Operators

Let's write more code in the same file:

```
const age = 22, country = 'Nigeria', isEnrolled = true;

if (country === 'Nigeria' && age >= 18) {
    console.log('Enrolled adult from Nigeria.');
}
if (country === 'Nigeria' || country === 'Ghana') {
    console.log('West African developer.');
}
if (!isEnrolled) {
    console.log('Not enrolled.');
}
```

Operator	Python Equivalent	Meaning
&&	and	Both sides must be true.
||	or	Either side being true is enough.
!	not	Flips true to false and false to true.

Since isEnrolled is true, !isEnrolled is false, so that last block never runs.

> **Be Careful Checking Truthiness With Numbers**
> *You can write if (name) as a quick check that name isn't empty, null, or undefined — this relies on truthy/falsy values. But be careful with numbers: if (age) treats 0 as false too. When 0 is a genuinely valid value, it's safer to compare it directly, like age !== null.*

3. Ternary Operator & switch

There is a shorthand way to write a simple if-else in one line, called the ternary operator — you may have already noticed it used inside a template literal in an earlier lesson:

```
const grade = score >= 70 ? 'A' : 'B';
console.log(grade);
```

This reads as: if score >= 70 is true, grade becomes 'A'; otherwise, grade becomes 'B'.

When comparing one variable against several specific values, switch is often cleaner than a long chain of if-else statements. Create a file named switch.js:

```
switch (day) {
    case 'Monday':
        console.log('JavaScript class today.');
        break;
    case 'Saturday':
    case 'Sunday':
        console.log('Weekend - rest and review.');
        break;
    default:
        console.log('Invalid day.');
}
```

> **Never Forget break**
> *Without break at the end of each case, JavaScript keeps running the code in the next case too — this is called falling through, and it usually isn't what you want. Notice Saturday and Sunday share the same result simply by being listed one after another with no break between them.*

4. Nullish Coalescing (??) and Optional Chaining (?.)

Two more operators come up often when working with real data, where some fields might be missing. Create a file named more_operators.js:

```
const age = 0;
console.log(age || 25);
console.log(age ?? 25);

const dev = { name: 'Kofi' };
console.log(dev?.contact?.phone);
console.log(dev?.contact?.phone ?? 'No phone on file');
```

Save and run:

Line	Result	Why
age || 25	25	|| treats 0 as falsy, so it falls back to the second value.
age ?? 25	0	?? only falls back when the value is exactly null or undefined — it leaves 0 alone, making it the safer choice when 0 could be a genuine value.
dev?.contact?.phone	undefined	dev has no contact property at all — optional chaining (?.) safely returns undefined instead of crashing, unlike dev.contact.phone directly, which would throw an error.
dev?.contact?.phone ?? 'No phone on file'	'No phone on file'	Combining ?. with ?? gives a clean fallback message for missing data.

Key Takeaways

if, else if, and else work exactly like Python's logic, with parentheses around conditions and curly brackets around each block. &&, ||, and ! are the direct equivalents of Python's and, or, and not. The ternary operator gives a one-line if-else, switch handles comparing

one value against many cases (never forgetting break), and ?? and ?. protect your code from missing or zero values crashing an expression. With decision-making covered, the next lesson moves into loops — the for loop, while, do...while, for...of, for...in, and the break and continue keywords that control them.