Return Values, Early Return & Callbacks

AiCE Software Engineering Foundations — JavaScript & Node.js

You know all three ways to define a function in JavaScript. This lesson covers what functions actually do once called — how they send values back, the professional pattern for handling errors early, and passing functions themselves as arguments to other functions.

1. Return Values & Early Return

A function sends a value back using return, and stops running the moment it hits one. If a function has no return statement, it simply gives back undefined.

```
const validateEmail = email => {
    if (!email) return { valid: false, error: 'Email is
required.' };
    if (!email.includes('@')) return { valid: false, error:
'Must contain @.' };
    return { valid: true, error: null };
};

const result = validateEmail('adesina@aice.africa');
console.log(result.valid ? 'Valid' : result.error);
```

Save and run — this checks the email step by step, and returns immediately as soon as it finds a problem, without checking anything further.

> **Early Return Is the Professional Standard**
> *Returning immediately once a problem is found — rather than nesting the rest of the logic inside an else — is called early return. It's the standard, professional way to write validation logic, keeping each check flat and readable instead of deeply nested.*

2. Functions as Arguments — Callbacks

In JavaScript, functions can be stored in variables and passed into other functions as arguments, just like any other value. A function passed this way is called a callback. You have actually already used this without knowing its name — forEach(), find(), and filter() all take a callback function.

```
const students = [
    { name: 'Adesina', score: 85 },
    { name: 'Fatima', score: 42 },
];

function processStudents(list, callback) {
    for (const student of list) callback(student);
}

const printPass = s => console.log(`${s.name}: ${s.score >= 50
? 'PASS' : 'FAIL'}`);

processStudents(students, printPass);
processStudents(students, s => console.log(s.name));
```

Save and run — the same processStudents function gives completely different output, depending on which callback is passed into it. That flexibility is the whole point of callbacks.

3. Practice: A Functions Showcase

Let's build a final project practicing everything from this lesson — a small showcase using all three ways of defining a function together. Create a file named showcase.js:

```
// AiCE Functions Showcase
function formatCurrency(amount) {
    return `₦${amount}`;
}

const getGrade = function (score) {
```

```
    return score >= 70 ? 'A' : score >= 50 ? 'C' : 'F';
};

const toTitleCase = str =>
    str.split(' ').map(w => w[0].toUpperCase() +
w.slice(1)).join(' ');

const students = [
    { name: 'adesina olagunju', score: 85 },
    { name: 'fatima zahra', score: 48 },
];

students.forEach(s => {
    console.log(`${toTitleCase(s.name)} | Grade:
${getGrade(s.score)} | ${formatCurrency(s.score * 500)}`);
});
```

Save and run — this brings together all three forms of a function: a declaration (formatCurrency), an expression (getGrade), and an arrow function (toTitleCase), working together in one program, exactly as you'll see in real JavaScript codebases. Push your work to GitHub.

Key Takeaways

A function returns a value with return and stops immediately once it does — early return keeps validation logic flat and readable, checking one condition at a time and exiting as soon as a problem is found. Functions are values in JavaScript, which means they can be passed into other functions as callbacks — the mechanism behind forEach(), find(), filter(), and any function that takes different behavior as an argument. With function declarations, expressions, arrow functions, default and rest parameters, return values, early return, and callbacks all covered, you now have everything you need to read and write JavaScript at the same level you already work in Python.