Declaring Functions — Declarations, Expressions & Arrow Functions

AiCE Software Engineering Foundations — JavaScript & Node.js

Conditionals and loops are covered — how a JavaScript program makes decisions and repeats actions. This final lesson covers functions — a block of code you write once and reuse, exactly like Python's def. JavaScript gives three different ways to define one, and by the end of this lesson, you'll know all three, and when to use each.

1. Function Declarations

Create a file named functions.js:

```
function greet(name) {
    return `Hello, ${name}! Welcome to AiCE.`;
}

console.log(greet('Adesina'));
console.log(greet('Kwame'));
```

This is called a function declaration, and it's the closest to Python's def — the function keyword, parameters in parentheses, and the body inside curly brackets.

> **Function Declarations Are Hoisted**
> *JavaScript moves function declarations to the top of the file behind the scenes, so you can actually call one before it appears in your code — this only happens with function declarations, not the other two forms in this lesson.*

```
console.log(getFullName('Adesina', 'Olagunju'));

function getFullName(firstName, lastName) {
    return `${firstName} ${lastName}`;
}
```

getFullName() was called before it was even defined, and it still worked — this is hoisting in action.

2. Function Expressions & Arrow Functions

Let's write more code in the same file:

```
const getGrade = function (score) {
    if (score >= 70) return 'A';
    if (score >= 50) return 'C';
    return 'F';
};

console.log(getGrade(85));
```

This second form is called a function expression — an unnamed function stored inside a variable. Unlike function declarations, function expressions are not hoisted, so you must define one before calling it.

The third form is the one you'll see used the most in modern JavaScript — the arrow function:

```
const getGrade = (score) => {
    if (score >= 70) return 'A';
    if (score >= 50) return 'C';
    return 'F';
};
```

This does exactly the same thing as the function expression above it, with shorter syntax. Arrow functions come with two shortcuts that make them even more concise:

```
const shout = name => `${name.toUpperCase()}!`;
const double = n => n * 2;
const isAdult = age => age >= 18;

console.log(shout('Africa'));
console.log(double(15));
```

```
console.log(isAdult(25));
```

Save and run — this gives 'AFRICA!', 30, and true.

Shortcut	Rule
Single parameter	Drop the parentheses around it — name instead of (name).
Single-expression body	Drop the curly brackets and the return keyword entirely — the value returns automatically.

3. Default & Rest Parameters

Just like Python, JavaScript functions can have default parameter values:

```
const registerStudent = (name, country, level = 'Beginner') =>
{
    return { name, country, level };
};

console.log(registerStudent('Adesina', 'Nigeria',
'Advanced'));
console.log(registerStudent('Amara', 'Senegal'));
```

Save and run — in the second call, no level was passed at all, so it fell back to 'Beginner' automatically.

There is also a rest parameter, using three dots before the parameter name. It collects any number of extra arguments into a single array — useful when you don't know in advance how many values will be passed in:

```
const calculateAverage = (...scores) => {
    const total = scores.reduce((sum, s) => sum + s, 0);
    return (total / scores.length).toFixed(2);
};
```

```
console.log(calculateAverage(85, 90, 78));
console.log(calculateAverage(72, 68, 75, 80));
```

calculateAverage() works whether it's passed three scores or four — the ...scores rest parameter gathers whatever is passed in into one array, worked with here using reduce(), a method that adds every item in an array together into a single total.

Key Takeaways

JavaScript has three ways to define a function: function declarations (hoisted, closest to Python's def), function expressions (stored in a variable, not hoisted), and arrow functions (the modern standard, with shortcuts for single parameters and single-expression bodies). Default parameters supply a fallback value when an argument is omitted, and rest parameters (...) gather any number of extra arguments into a single array. With three ways to define a function covered, the next lesson finishes this unit with what functions actually do once called — return values, early return, and passing functions themselves as arguments to other functions, called callbacks.