# Optional chaining

Optional chaining (represented by the `?.` operator) is a feature in several programming languages, including JavaScript (introduced in ES2020), TypeScript, and Swift. It provides a concise and safe way to access properties or call methods on an object when there's a possibility that an intermediate property in the chain might be `null` or `undefined`.

How it Works:

When the optional chaining operator `?.` is used to access a property or call a method, the expression evaluates as follows:

- **Check for Nullish Values:**

  The operator checks if the value immediately before `?.` is `null` or `undefined`.

- **Short-Circuiting:**

  If the value is `null` or `undefined`, the entire expression immediately short-circuits and evaluates to `undefined`, preventing a `TypeError` from being thrown.

- **Continue Chaining:**
  If the value is neither `null` nor `undefined`, the property access or method call proceeds as normal, and the chain continues to the next part of the expression.

Benefits:

- **Cleaner Code:**

  It eliminates the need for verbose `if` statements or ternary operators to check for the existence of nested properties, making the code more readable and compact.

- **Error Prevention:**
  It gracefully handles situations where properties or methods might be missing, preventing runtime errors like "Cannot read property 'x' of undefined."

Examples in JavaScript:

JavaScript

```javascript
const user = {
  name: "Alice",
  address: {
    street: "123 Main St",
    city: "Anytown"
  }
};

// Accessing a nested property with optional chaining
console.log(user.address?.city); // Output: Anytown

const newUser = {
  name: "Bob"
};

// Accessing a missing nested property with optional chaining
console.log(newUser.address?.street); // Output: undefined (no error)

// Calling a method with optional chaining
const obj = {
  method: () => "Hello!"
};
console.log(obj.method?.()); // Output: Hello!

const anotherObj = {};
console.log(anotherObj.method?.()); // Output: undefined (no error)
```