Absolutely, Tom! JavaScript has quite a few quirky ambiguities that can trip up developers. Here are some interesting ones:

### **1. `typeof null` Returns `"object"`**
You might expect `typeof null` to return `"null"`, but instead:
```javascript
console.log(typeof null); // "object"
```
This is a long-standing bug in JavaScript that has never been fixed for backward compatibility.

---

### **2. `NaN` ("Not a Number") is Actually a Number**
Despite its name, `NaN` is considered a numeric type:
```javascript
console.log(typeof NaN); // "number"
```
Even more bizarre, `NaN` is **not** equal to itself:
```javascript
console.log(NaN === NaN); // false
```
Which is why we use `Number.isNaN()` to check for `NaN`.

---

### **3. `[] + []` Returns an Empty String**
Adding two empty arrays results in an empty string, instead of something more intuitive like `[[], []]`:
```javascript
console.log([] + []); // ""
```
This happens because JavaScript tries to convert objects into strings before performing addition.

---

### **4. `true + true` Equals `2`**
Adding two boolean values gives an unexpected numerical result:
```javascript
console.log(true + true); // 2
```
This happens because JavaScript treats `true` as `1` and `false` as `0` in arithmetic operations.

---

### **5. `0.1 + 0.2 !== 0.3`**
Floating point precision errors mean simple math can be slightly off:
```javascript
console.log(0.1 + 0.2 === 0.3); // false
console.log(0.1 + 0.2); // 0.30000000000000004
```
This happens because JavaScript uses **binary floating-point arithmetic**, which can't precisely represent certain decimal values.

---

### **6. Implicit Type Coercion Can Lead to Weird Results**
JavaScript tries to convert types automatically, sometimes in surprising ways:
```javascript
console.log(1 + "2"); // "12" (string concatenation)
console.log("5" - 2); // 3 (subtraction forces numeric conversion)
console.log(false == 0); // true (false is treated as 0)
console.log("" == 0); // true (empty string is treated as 0)
```
This can lead to unexpected behavior if you're not careful.

---

JavaScript's flexibility is both a strength and a source of confusion! Do any of these stand out to you? 😄