Errors

Ref Javascript

Reading error messages

When JavaScript cannot carry on, it stops and shows an error . The first word names the kind of error; the rest says what went wrong and where. Read it, then look at the line it names.

```
SyntaxError: missing ) after argument list
ReferenceError: totl is not defined
TypeError: Cannot read properties of undefined (reading 'length')
```

Error	What it means	A typical cause
`SyntaxError`	the code is not written correctly, so nothing runs	a missing bracket or quote
`ReferenceError`	a name that does not exist	a spelling mistake : `totl` for `total`
`TypeError`	the value is the wrong type for what you did	using `.length` on `undefined`

The exact wording can differ between browsers, but the name of the error is the same. This program catches each error so that it can print its name:

```
try {
  const total = 5;
  console.log(totl);
} catch (e) {
  console.log(e.name);           // ReferenceError
}

try {
  const pupil = undefined;
  console.log(pupil.length);
} catch (e) {
  console.log(e.name);           // TypeError
}
```

Some mistakes give no error at all, just a wrong answer: a logic error . To find one, print the values along the way with `console.log`, and check them by hand.

try, catch & throw

Code that might fail goes in a `try` block. If an error happens there, JavaScript jumps to `catch` instead of stopping, and `e` holds the error:

```
try {
  JSON.parse("{not valid}");
  console.log("parsed");          // skipped
} catch (e) {
  console.log("Could not read the data");
}
console.log("The program carries on");
```

Your own code can raise an error with `throw`. This lets a function refuse bad input clearly, instead of returning a wrong answer:

```
function average(list) {
  if (list.length === 0) {
    throw new Error("the list is empty");
  }
  return list.reduce((s, x) => s + x, 0) / list.length;
}

console.log(average([4, 6]));    // 5
try {
  average([]);
} catch (e) {
  console.log(`Error: ${e.message}`);   // Error: the list is empty
}
```

- `new Error("…")` makes an error whose `message` is your text.
- Only use `try` around code that can really fail, such as reading data. Do not use it to hide mistakes in your own code.

Common mistakes

- Skipping the message. Its **first** line names the kind of error and what went wrong; start there.
- A `catch` that does nothing (`catch (e) {}`) hides the problem: at least print `e.message`.
- `throw "text"` works, but `throw new Error("text")` also records where the error happened.