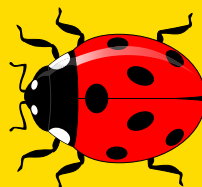


Debugger in JS

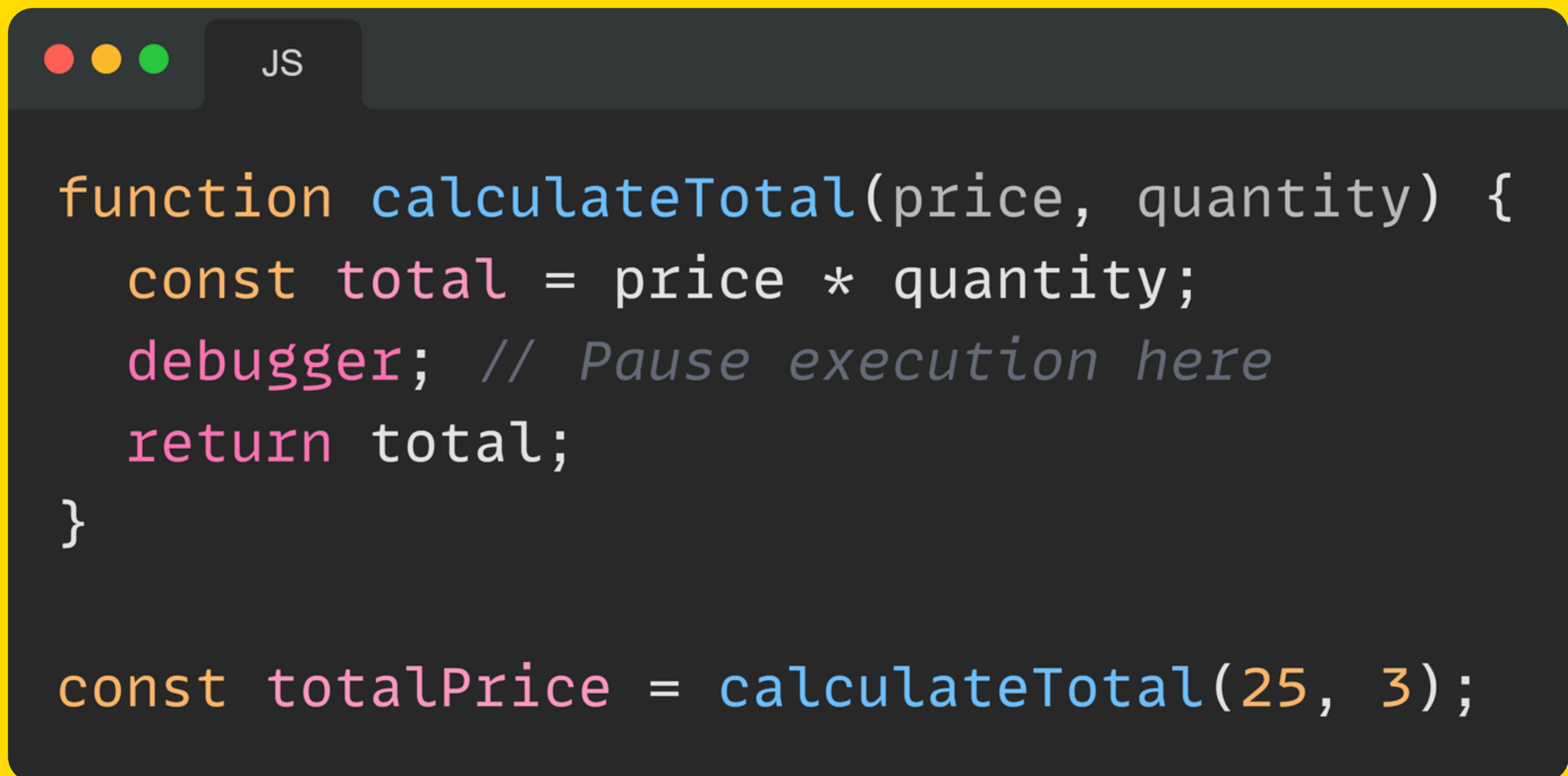


www.scribbler.live



What is debugger ?

- A statement in JavaScript that triggers a breakpoint in your code, allowing you to pause execution and inspect variables, track function calls, and navigate through code step-by-step.

A code editor window with a dark theme. The top bar shows a tab labeled 'JS' with three colored window control buttons (red, yellow, green) to its left. The code is written in a light-colored font. A line of code is highlighted in blue, indicating a breakpoint. The code is as follows:

```
function calculateTotal(price, quantity) {  
  const total = price * quantity;  
  debugger; // Pause execution here  
  return total;  
}  
  
const totalPrice = calculateTotal(25, 3);
```



6 Things to Know



Setting Breakpoints

Inspecting Variables

Stepping Through Code

Conditional Breakpoints

Logging and Beyond

Cross-Browser Compatibility

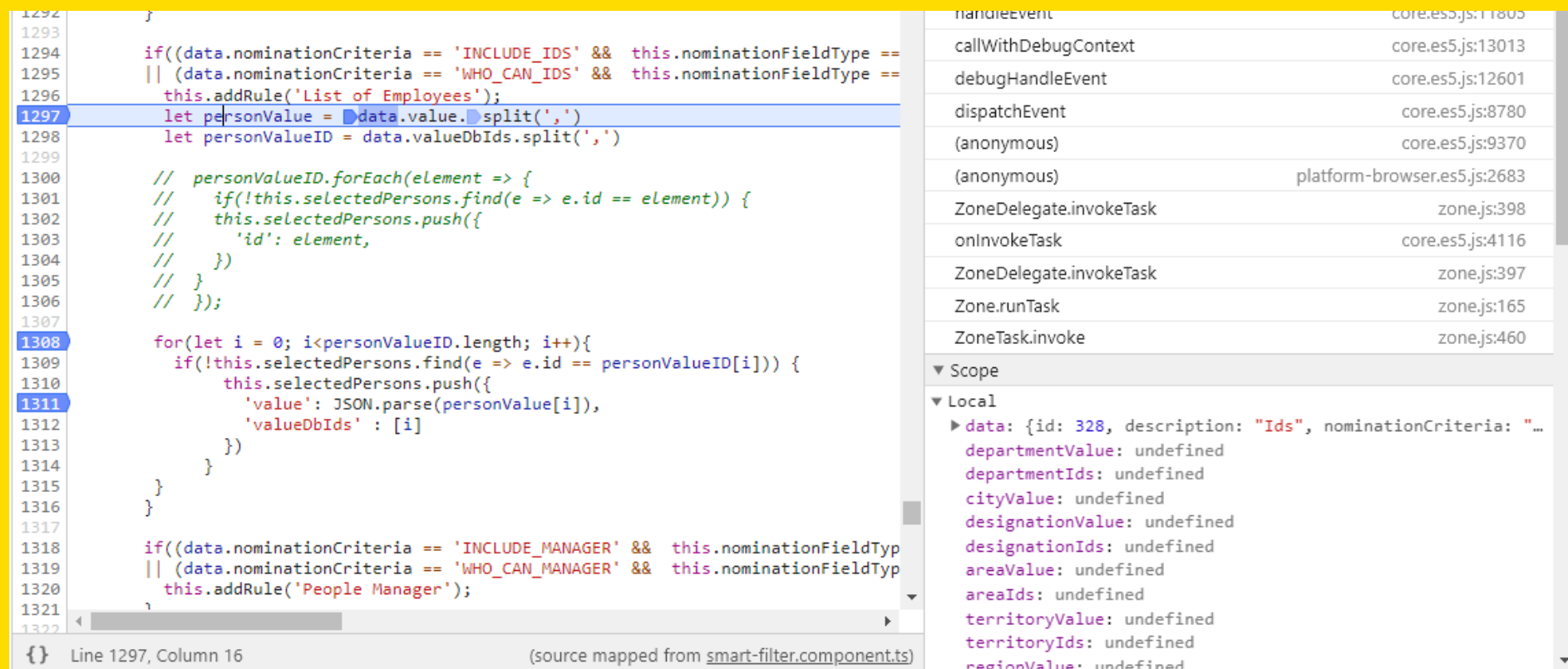
Setting Breakpoints

- Place debugger statements strategically at points in your code where you suspect issues or want to inspect variables' values.



Inspecting Variables

- Once paused at a debugger breakpoint, utilize your browser's developer tools to inspect the current state of variables, parameters, and the call stack.



```
1292 }
1293
1294 if((data.nominationCriteria == 'INCLUDE_IDS' && this.nominationFieldType ==
1295 || (data.nominationCriteria == 'WHO_CAN_IDS' && this.nominationFieldType ==
1296 this.addRule('List of Employees');
1297 let personValue = data.value.split(',')
1298 let personValueID = data.valueDbIds.split(',')
1299
1300 // personValueID.forEach(element => {
1301 //   if(!this.selectedPersons.find(e => e.id == element)) {
1302 //     this.selectedPersons.push({
1303 //       'id': element,
1304 //     })
1305 //   }
1306 // });
1307
1308 for(let i = 0; i<personValueID.length; i++){
1309   if(!this.selectedPersons.find(e => e.id == personValueID[i])) {
1310     this.selectedPersons.push({
1311       'value': JSON.parse(personValue[i]),
1312       'valueDbIds' : [i]
1313     })
1314   }
1315 }
1316
1317
1318 if((data.nominationCriteria == 'INCLUDE_MANAGER' && this.nominationFieldType
1319 || (data.nominationCriteria == 'WHO_CAN_MANAGER' && this.nominationFieldType
1320 this.addRule('People Manager');
1321
1322
```

Line 1297, Column 16 (source mapped from smart-filter.component.ts)

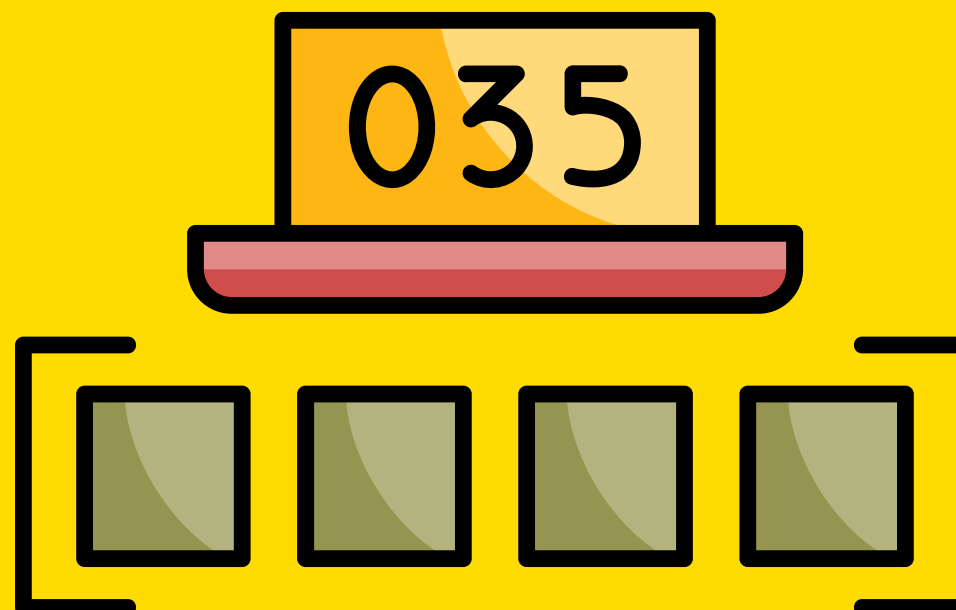
Call Stack:

- handleEvent core.es5.js:11805
- callWithDebugContext core.es5.js:13013
- debugHandleEvent core.es5.js:12601
- dispatchEvent core.es5.js:8780
- (anonymous) core.es5.js:9370
- (anonymous) platform-browser.es5.js:2683
- ZoneDelegate.invokeTask zone.js:398
- onInvokeTask core.es5.js:4116
- ZoneDelegate.invokeTask zone.js:397
- Zone.runTask zone.js:165
- ZoneTask.invoke zone.js:460

Scope:

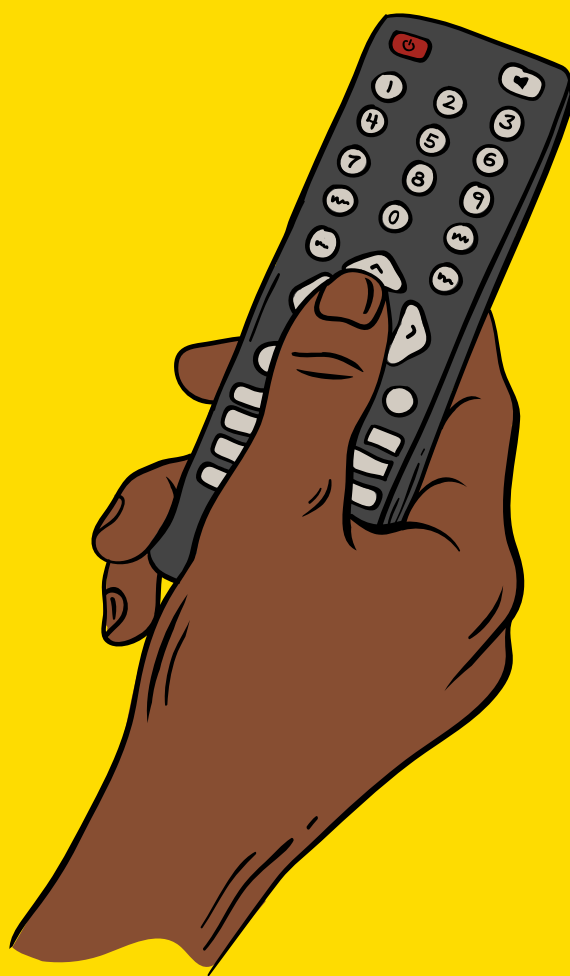
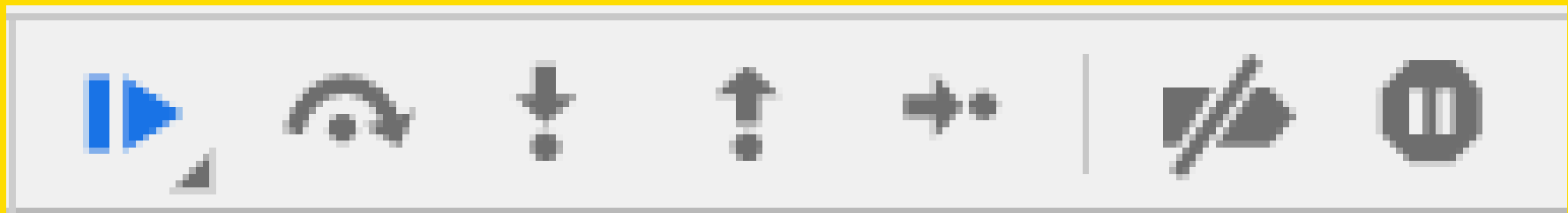
Local:

- data: {id: 328, description: "Ids", nominationCriteria: "...", departmentValue: undefined, departmentIds: undefined, cityValue: undefined, designationValue: undefined, designationIds: undefined, areaValue: undefined, areaIds: undefined, territoryValue: undefined, territoryIds: undefined, regionValue: undefined}



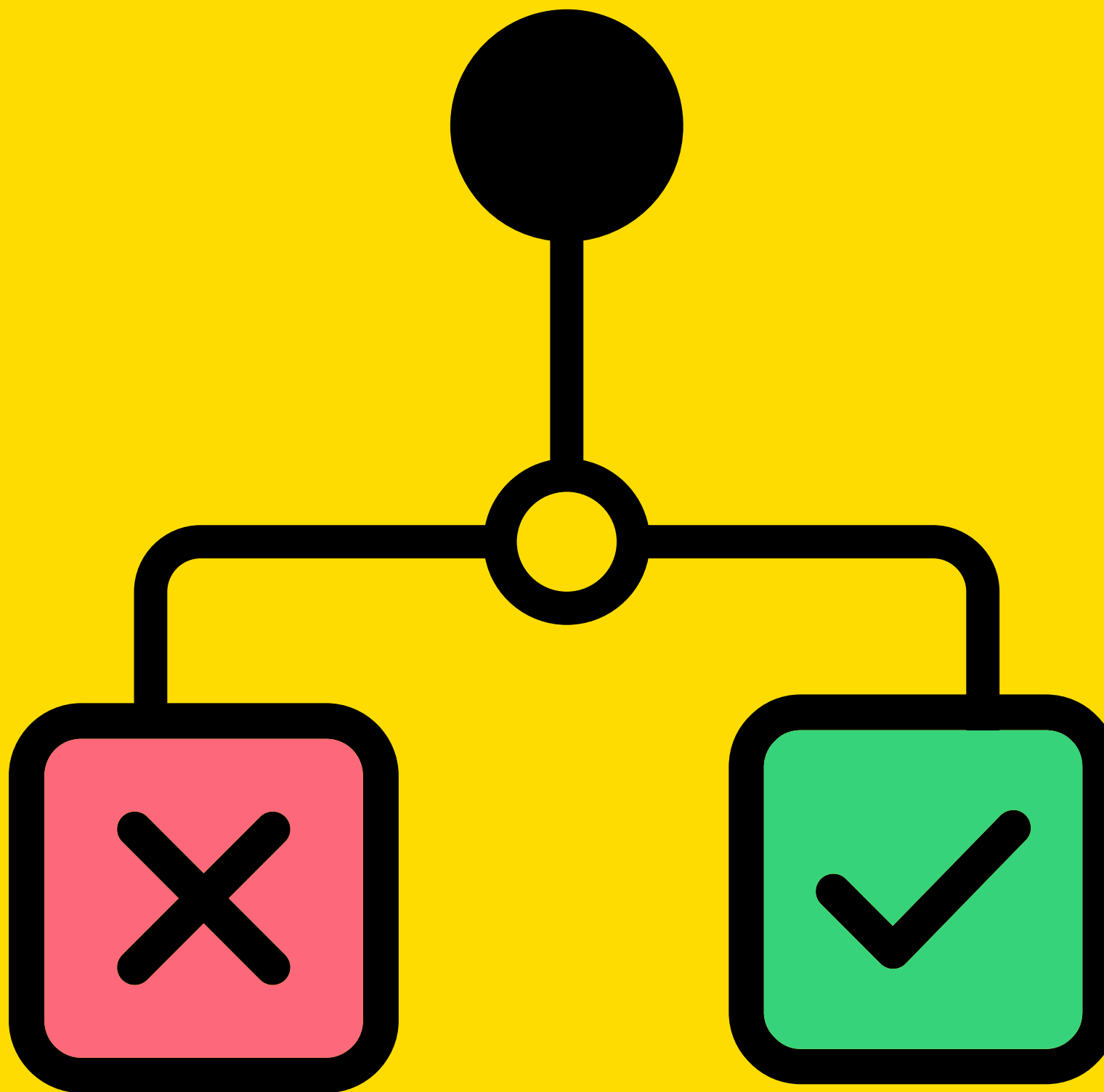
Stepping Through Code

- Use step-by-step execution controls (such as Step Into, Step Over, and Step Out) provided by your debugger to traverse through code and understand its behavior.



Conditional Breakpoints

- Enhance your debugging efficiency by setting conditional breakpoints, which trigger only when specified conditions are met.



Logging and Beyond

- While `console.log()` is invaluable, debugger offers real-time insights into code execution, allowing you to intervene and analyze complex scenarios effectively.



Cross-Browser Compatibility

- debugger is supported across major browsers and developer tools, ensuring consistent debugging experiences regardless of your development environment.



Keep Exploring Javascript with us!

Share this with a friend who needs it and
make sure to practice these in scribbler.



Scribbler.live

Free and Open Interface to
experiment JavaScript