

EECS 368

Programming Language Paradigms

David O. Johnson

Fall 2022

Reminders

- Assignment 2 due: 11:59 PM, Monday, September 19
- Assignment 3 due: 11:59 PM, Monday, October 3

Any Questions?

In-Class Problem Solution

- 7-(9-9) In-Class Problem Solution.pptx

Any Questions?

Bugs & Debugging

- Flaws in computer programs are usually called **bugs**.
- The process of finding mistakes—bugs—in programs is called **debugging**.

“Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

Brian Kernighan and P.J. Plauger, *The Elements of Programming Style*

JavaScript Makes Debugging Hard

- JavaScript's looseness makes debugging hard.
- Its concept of bindings and properties is vague enough that it will rarely catch typos before actually running the program.
- And even then, it allows you to do some clearly nonsensical things without complaint, such as computing: `true * "monkey"`.
- Often, your nonsense computation will merely produce NaN (not a number) or an undefined value, while the program happily continues, convinced that it's doing something meaningful.
- The mistake will manifest itself only later, after the bogus value has traveled through several functions.
- It might not trigger an error at all but silently cause the program's output to be wrong.
- Finding the source of such problems can be difficult.
- Fortunately, JavaScript does have some tools to help.

Errors that JavaScript Does Complain About

- Syntax errors
 - Writing a program that does not follow JavaScript's syntax will immediately make the computer complain.
- Other things that will cause an error to be reported when the program tries to perform the action include:
 - Calling something that's not a function
 - Looking up a property on an undefined value

Strict Mode

- JavaScript can be made a little stricter by enabling **strict mode**.
- This is done by putting the string **"use strict"** at the top of a file or a function body.

```
function canYouSpotTheProblem() {  
  "use strict";  
  for (counter = 0; counter < 10; counter++) {  
    console.log("Happy happy");  
  }  
}  
  
canYouSpotTheProblem();  
// → ReferenceError: counter is not defined
```

- Normally, when you forget to put **let** in front of your binding, as with **counter** in the example, JavaScript quietly creates a global binding and uses that.
- In strict mode, an error is reported instead.
- Strict mode does a few other things.
- In short, putting **"use strict"** at the top of your program rarely hurts and might help you spot a problem.

Types

- Some languages want to know the types of all your bindings and expressions before even running a program.
- They will tell you right away when a type is used in an inconsistent way.
- JavaScript considers types only when actually running the program, and even then often tries to implicitly convert values to the type it expects, so it's not much help.
- Still, types provide a useful framework for talking about programs.
- A lot of mistakes come from being confused about the kind of value that goes into or comes out of a function.
- If you have that information written down, you're less likely to get confused.
- You could add a comment like the following to describe its type:

```
// (state, memory) → {state: string, memory: Array}  
function goalOrientedRobot(state, memory) {  
  // ...  
}
```
- There are a number of different conventions for annotating JavaScript programs with types.

Any Questions?

Testing

- Computers are good at repetitive tasks, and testing is the ideal repetitive task.
- Automated testing is the process of writing a program that tests another program.
- Writing tests is a bit more work than testing manually.
- But once you've done it, it takes you only a few seconds to verify that your program still behaves properly in all the situations you wrote tests for.
- Tests usually take the form of little labeled programs that verify some aspect of your code.
- For example, a set of tests for the `toUpperCase` method might look like this:

```
function test(label, body) {  
  if (!body()) console.log(`Failed: ${label}`);  
}
```

```
test("convert Latin text to uppercase", () => {  
  return "hello".toUpperCase() == "HELLO";  
});
```

Any Questions?

Debugging with console.log

- Putting a few strategic `console.log` calls into the program is a good way to get additional information about what the program is doing.
- Printing the content of variables is an effective way to debug in any language
- The following example program tries to convert a whole number to a string in a given base (decimal, binary, and so on) by repeatedly picking out the last digit and then dividing the number to get rid of this digit.
- But the strange output that it currently produces suggests that it has a bug.

```
function numberToString(n, base = 10) {  
  let result = "", sign = "";  
  if (n < 0) {  
    sign = "-";  
    n = -n;  
  }  
  do {  
    result = String(n % base) + result;  
    n /= base;  
  } while (n > 0);  
  return sign + result;  
}  
console.log(numberToString(13, 10));  
// → 1.5e-3231.3e-3221.3e-3211.3e-  
3201.3e-3191.3e-3181.3...
```

Debugging with console.log

- Putting a few strategic `console.log` calls into the program is a good way to get additional information about what the program is doing.
- In this case, we want `n` to take the values 13, 1, and then 0. Let's write out its value at the start of the loop.
13
1.3
0.13
0.013
...
1.5e-323
- Dividing 13 by 10 does not produce a whole number.
- Instead of `n /= base`, what we actually want is `n = Math.floor(n / base)` so that the number is properly "shifted" to the right.

```
function numberToString(n, base = 10) {  
  let result = "", sign = "";  
  if (n < 0) {  
    sign = "-";  
    n = -n;  
  }  
  do {  
    console.log(n);  
    result = String(n % base) + result;  
    n /= base;  
  } while (n > 0);  
  return sign + result;  
}  
  
console.log(numberToString(13, 10));  
// → 1.5e-3231.3e-3221.3e-3211.3e-  
3201.3e-3191.3e-3181.3...
```

Debugging with Browser Debugger

- An alternative to using `console.log` to peek into the program's behavior is to use the debugger capabilities of your browser.
- Browsers come with the ability to set a breakpoint on a specific line of your code.
- When the execution of the program reaches a line with a breakpoint, it is paused, and you can inspect the values of bindings at that point.
- Debuggers differ from browser to browser, but look in your browser's developer tools or search the Web for more information.
- Another way to set a breakpoint is to include a **debugger** statement (consisting of simply that keyword) in your program.
- If the developer tools of your browser are active, the program will pause whenever it reaches such a statement.

Any Questions?

Exception Handling

- When a function **cannot proceed** normally, ...
- ... what we would like to do is just **stop** what we are doing
- ... and immediately **jump** to a place that knows how to handle the problem.
- This is what **exception handling** does.
- Exceptions are a mechanism that makes it possible for code that runs into a problem to **throw** an exception.
- Throwing an exception causes the program to stop what it is doing and immediately jump to a place that knows how to handle the problem.
- Jumping to the place that knows how to handle the problem is called **catching** the exception.

Exception Handling

- The throw keyword is used to **throw** an exception.
- Catching one is done by wrapping a piece of code in a **try block**, followed by the **keyword catch**.
- When the code in the try block causes an exception to be raised, the catch block is evaluated, with the name in parentheses bound to the exception value.
- After the catch block finishes ...
- ... or if the try block finishes without problems
- ... the program proceeds beneath the entire try/catch statement.

```
function promptDirection(question) {  
  let result = prompt(question);  
  if (result.toLowerCase() == "left") return "L";  
  if (result.toLowerCase() == "right") return "R";  
  throw new Error("Invalid direction: " + result);  
}  
  
function look() {  
  if (promptDirection("Which way?") == "L") {  
    return "a house";  
  } else {  
    return "two angry bears";  
  }  
}  
  
try {  
  console.log("You see", look());  
} catch (error) {  
  console.log("Something went wrong: " + error);  
}
```

Exception Handling

- Running this program will result in the prompt:

⊕ This page says

Which way?

down|

OK

Cancel

- If you type “down” and click on OK, it will produce this output:

Something is wrong: Error: Invalid direction: down

```
function promptDirection(question) {  
  let result = prompt(question); 3  
  if (result.toLowerCase() == "left") return "L"; 4  
  if (result.toLowerCase() == "right") return "R"; 5  
  throw new Error("Invalid direction: " + result); 6  
}
```

```
function look() {  
  if (promptDirection("Which way?") == "L") { 2  
    return "a house";  
  } else {  
    return "two angry bears";  
  }  
}
```

```
try {  
  console.log("You see", look()); 1  
} catch (error) {  
  console.log("Something is wrong: " + error); 7  
}
```

Any Questions?

Assertions

- Assertions are checks inside a program that verify that something is the way it is supposed to be.
- They are used to find programmer mistakes, not to handle situations that can come up in normal operation.
- Some languages (e.g., Python, C++) allow assertions to be used during debugging and turned off by the compiler or interpreter once the program is error-free.
- JavaScript does not support assertions per se, but they can be manually added to code.
- If, for example, `firstElement` is described as a function that should never be called on empty arrays, we might write it like this:

```
function firstElement(array) {  
    //beginning of assertion  
    if (array.length == 0) {  
        throw new Error("firstElement called with []");  
    }  
    //end of assertion  
    return array[0];  
}
```

Assertions

- Now, instead of silently returning undefined ...
- ... which you get when reading an array property that does not exist
- ... this will loudly blow up your program as soon as you misuse it.
- This makes it less likely for such mistakes to go unnoticed and easier to find their cause when they occur.

```
function firstElement(array) {  
    //beginning of assertion  
    if (array.length == 0) {  
        throw new Error("firstElement called with []");  
    }  
    //end of assertion  
    return array[0];  
}
```

Summary

- Flaws in computer programs are usually called **bugs**.
- The process of finding mistakes—bugs—in programs is called **debugging**.
- JavaScript's looseness makes debugging hard.
- JavaScript does complain about:
 - Syntax errors
 - Calling something that's not a function
 - Looking up a property on an undefined value
- JavaScript can be made a little stricter by enabling **strict mode**.
- **Writing test programs** is a way to verify that your program still behaves properly in all the situations you wrote tests for.
- **Printing out values of key variables** with `console.log` is an effective means of debugging.
- **Using the debugger** capabilities of your browser is an alternative to using `console.log` to peek into the program's behavior.
- **Exception handling** with the keywords **throw**, **try**, and **catch** causes the program to stop what it is doing and immediately jump to a place that knows how to handle the problem.
- The keywords **throw**, **try**, and **catch** can be used to implement **assertions** in JavaScript.

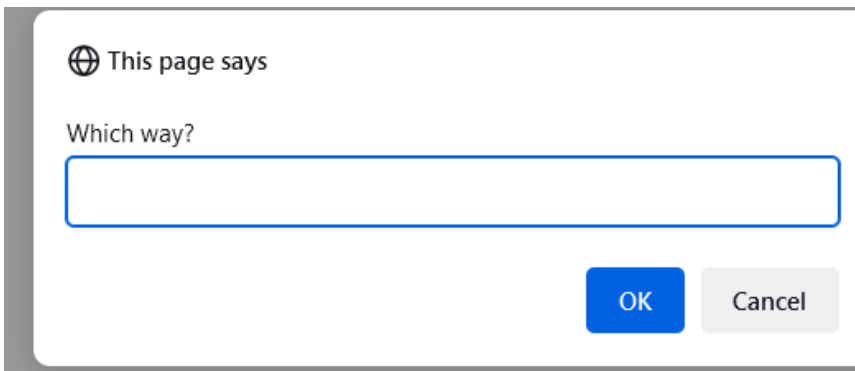
Any Questions?

In-Class Problem

What is the console output of this program, if you type:

1. Left
2. RIGHT
3. UP
4. List in order the statements executed by JavaScript for No. 1 above.
5. List in order the statements executed by JavaScript for No. 2 above.
6. List in order the statements executed by JavaScript for No. 3 above.

```
function promptDirection(question) {  
  let answer = prompt(question);  
  if (answer.toLowerCase() == "left") return "L";  
  if (answer.toLowerCase() == "right") return "R";  
  throw new Error("Invalid answer: " + answer);  
}  
  
function look() {  
  if (promptDirection("Which way?") == "L") {  
    return "a house";  
  } else {  
    return "two angry bears";  
  }  
}  
  
try {  
  console.log("You see", look());  
} catch (error) {  
  console.log("Something's not right: " + error);  
}
```



⊕ This page says

Which way?

OK Cancel