INLS 672
Web Development 2

JavaScript Debugging

UNC
SCHOOL OF INFORMATION
AND LIBRARY SCIENCE

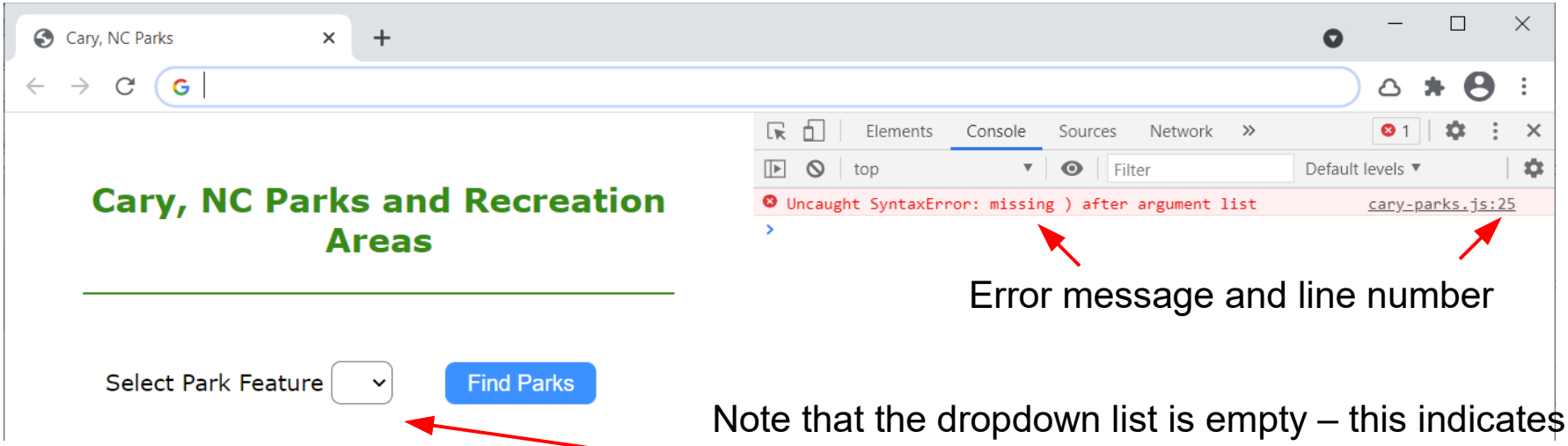
Joan Boone
jpboone@email.unc.edu

Using JavaScript `console.log()`

Simplest approach: use the Console

- Add `console.log()` statements to your code to display the values of important variables; similar to print() statements in Python
- View the `console.log()` output by Inspecting the page and viewing the Console
- If you load your page and the browser window is empty, you most likely have a JavaScript error. Check the Console tab – this is where error messages are logged, and typically include a line number in your code where the problem was detected.

Sample output when a syntax error occurs:

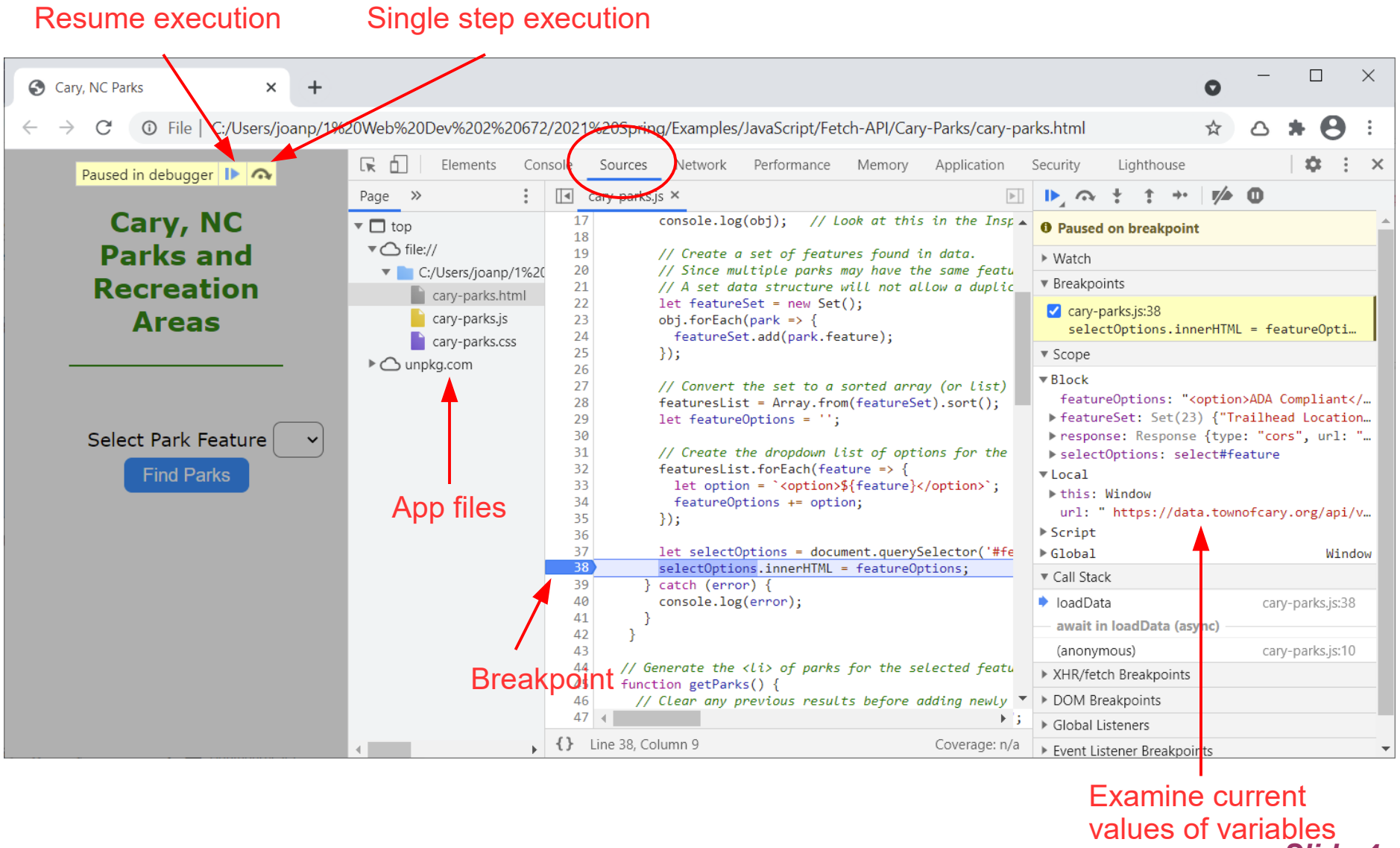


Using JavaScript Debugger Tools

Best approach for runtime and logic problems

- Both Chrome and Firefox browsers have built-in debugging tools that allow you to step through JavaScript code and examine its state to help track down problems.
  - Chrome: Debugging in Chrome
  - Firefox: JavaScript Debugger
- In general, for both browsers, the workflow is
  - Inspect the page, and select the tab that displays the Source code. This is 'Sources' in Chrome, and 'Debugger' in Firefox (as of this writing)
  - Select your JavaScript file to view the contents
  - You can set breakpoints in your code where you would like it to pause, so you can examine the current state of execution and variables. You set the breakpoint by clicking the line number in the code where you want it to pause, and the line number changes color to highlight its status; click it again to remove it.
  - Run your code again and it will pause at the breakpoint so you can examine your variables.
  - You can then resume your code, single step by executing one line at a time.
  - It is recommended that you read the browser documentation as they have very good, and detailed, descriptions of how to use the debuggers.

Using Chrome Debugger



Cross-Origin Resource Sharing (CORS) Errors

- A CORS policy allows servers to define whether origins (domains, schemes, ports) other than their own can request resources. A very simple example is a server in domain A can prohibit a client in domain B from requesting resources.
- For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts.
- For open data sources, such as those on Data.gov, cross-origin requests are usually allowed, so you will not encounter errors; however, in the example below, the JavaScript attempted to fetch the JSON contents at an Apple music site and was blocked by the server's CORS policy.
- If you encounter this kind of error, there is little you can do to work around it.