Debugging JavaScript and TypeScript

19/02/2024

Stephan Brommer Gutiérrez
alu0101493497@ull.edu.es

Tania Évora Vargas Martínez
alu0101219622@ull.edu.es

Outline

1. Introduction
2. Debugging in VSC
3. Debugging JavaScript (VSC)
4. Debugging TypeScript (VSC)
5. Debugging in the Browser
6. Debugging JavaScript (Browser)
7. Debugging TypeScript (Browser)
8. Bibliography and references

Introduction

Introduction

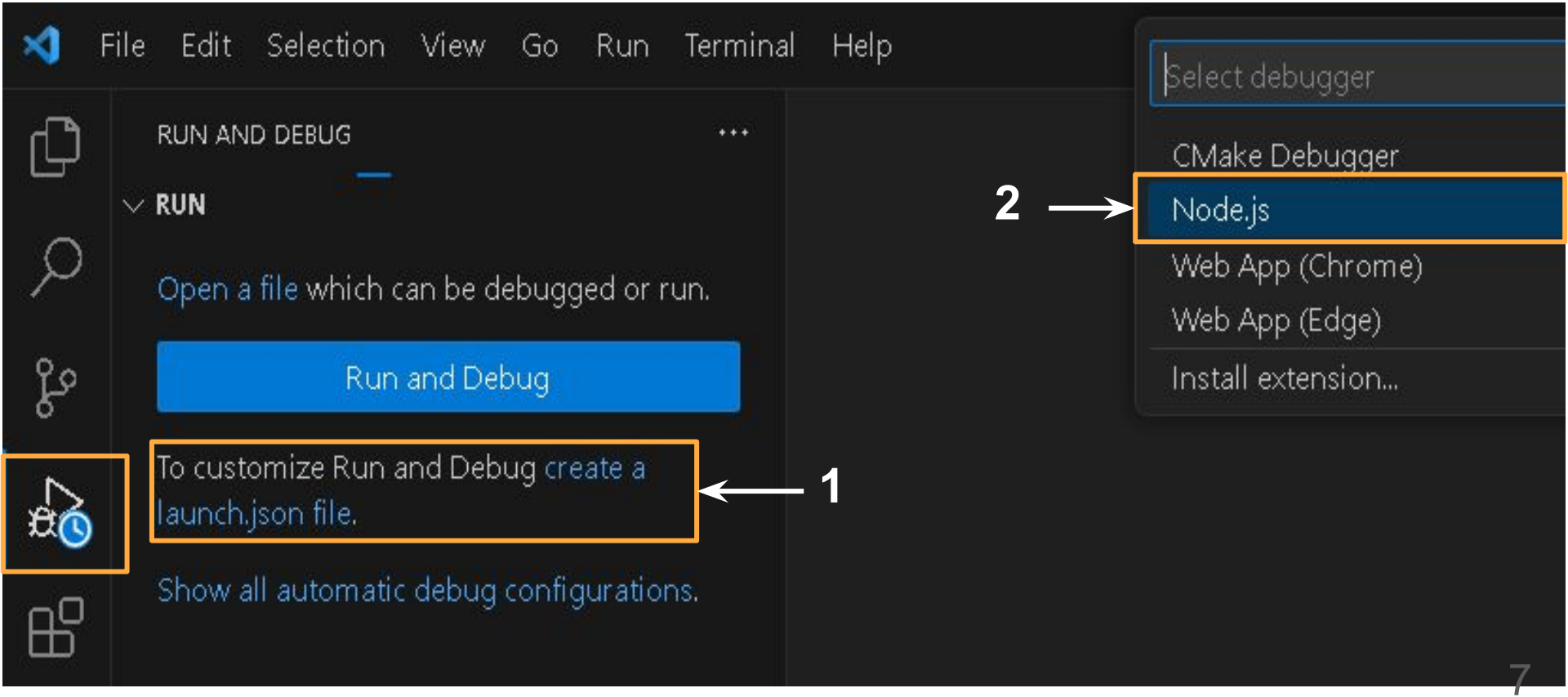
Unit Testing: We know there is a bug...

Programmer: But, where is it?

```
console.log()
```

Debugging in VSC

First steps


File Edit Selection View Go Run Terminal Help
RUN AND DEBUG
RUN
Open a file which can be debugged or run.
Run and Debug
To customize Run and Debug create a launch.json file.
Show all automatic debug configurations.
1
2
Select debugger
CMake Debugger
Node.js
Web App (Chrome)
Web App (Edge)
Install extension...

launch.json configuration

.vscode > {} launch.json > ...

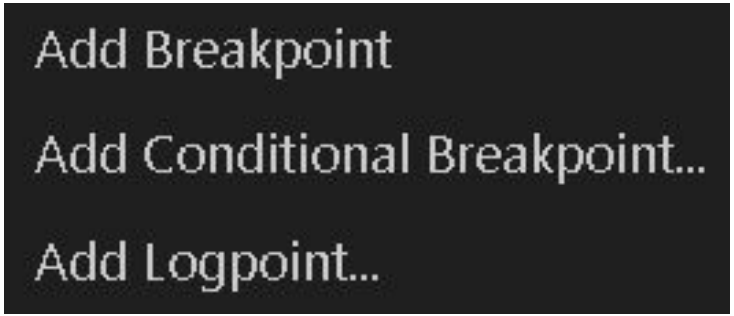
```jsonc
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Descriptive name",
            "program": "${workspaceFolder}/example.js"
        }
    ]
}
```

launch.json configuration

- **version:** launch.json version.
- **configurations:** Array of configurations, each one include
  - **type:** Debugger type ➡ node (JS/TS)
  - **request:** How the debugging starts ➡ launch
  - **name:** Descriptive name for the specific configuration
  - **program:** Path of the program to debug

Note: these attributes are mandatory

Breakpoints types



- **Breakpoint:** pause debugging
- **Logpoint:** console message
- **Conditional breakpoint:** pause when condition is true
  - **Hit point:** pause when reaches a value

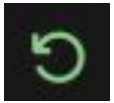
Debugger buttons


Continue until the next point


Run the following method without inspecting (unless there are breakpoints).


Execute the following method entering to inspect

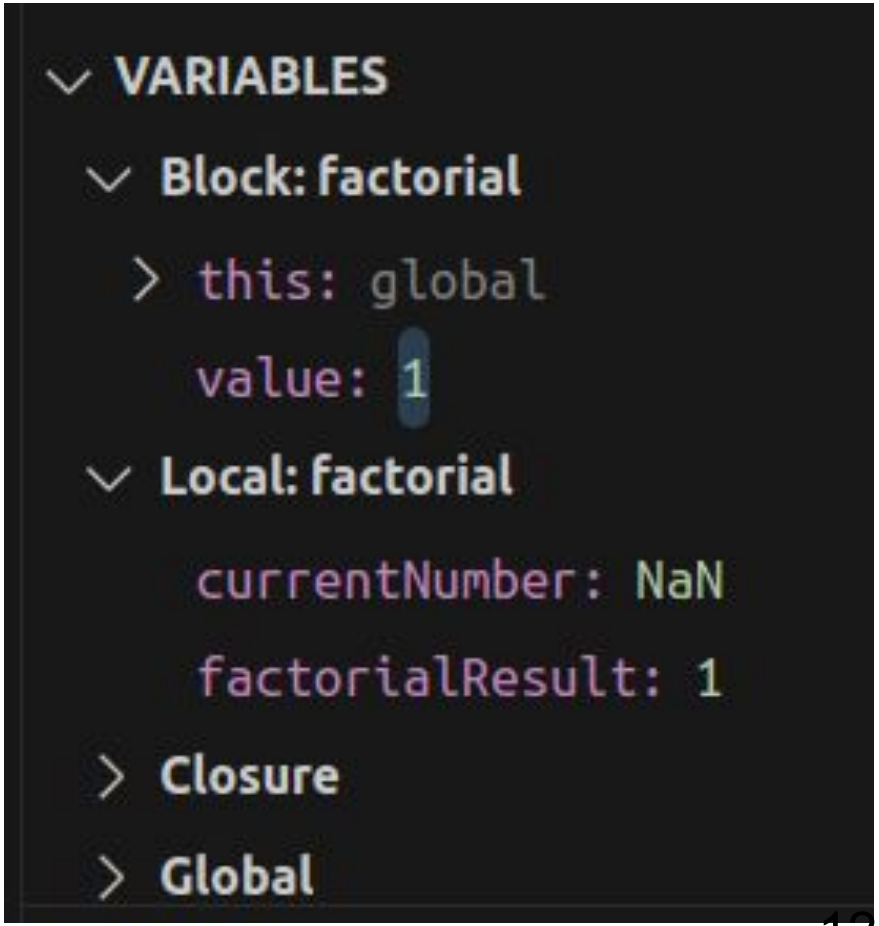

If we are inside a method, the execution completes without going line by line


Terminates the current program and re-executes it


Finish debugging

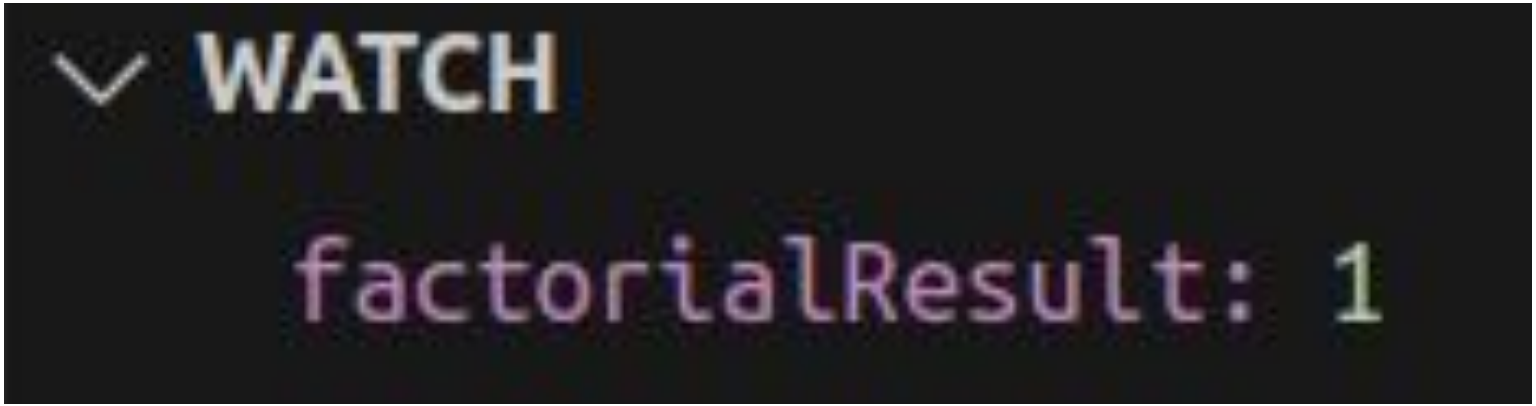
Panel: VARIABLES

- Track local values, of the current instance
- Track global variables
- Closures: nested function variables



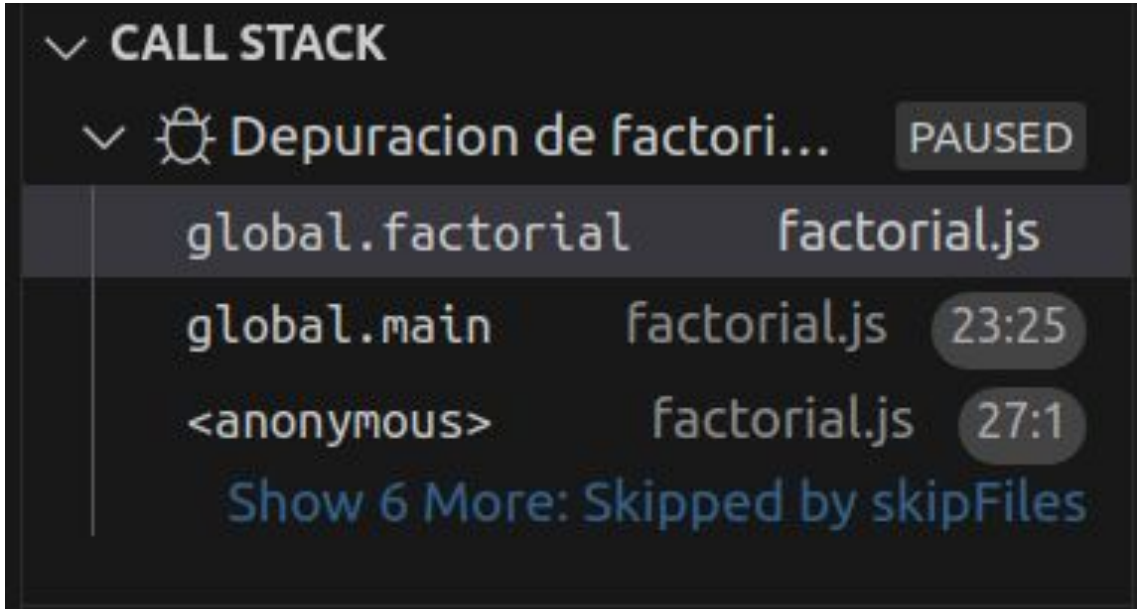
Panel: WATCH

- Monitor and observe the value of a specific expression or variable during program execution.



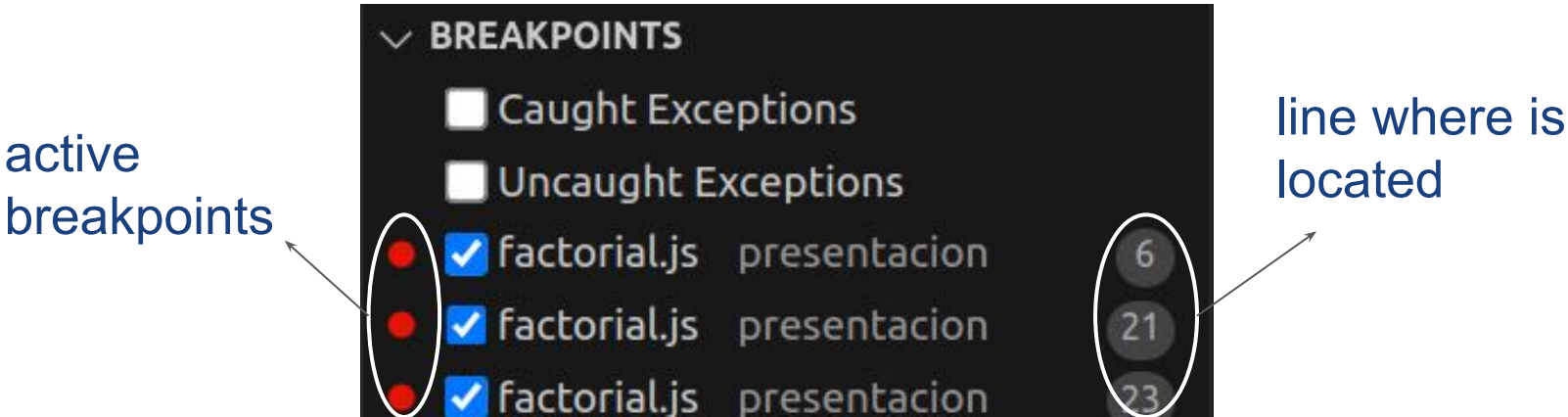
Panel: CALL STACK

- To see the order of calling functions and methods



Panel: BREAKPOINTS

- Modification and manipulation of breakpoints
- Quick access to the line where you are
- **Caught Exceptions:** Pause if an exception occurs
- **Uncaught Exceptions:** Ignore exceptions that occur



Debugging in JavaScript (VSC)

Repository example

https://github.com/ULL-ESIT-PAI-2023-2024/2023-2024-pai-debugging-stephanBG-taniaVM/tree/master/src/vsc/JavaScript

Debugging in TypeScript (VSC)

First steps

1. Create tsconfig.json
2. Create the launch.json
3. Compile with tsc, from wherever tsconfig.json is located
4. Start debugging

tsconfig.json > ...

```
{
  "compilerOptions": {
    "target": "ES6",
    "module": "CommonJS",
    "strict": true,
    "sourceMap": true
  },
  "include": [
    "./**/*.ts"
  ]
}
```

tsconfig.json configuration

tsconfig.json configuration

- **compilerOptions**: Compile options.
  - **target:** Version of code that we want it to generate (ES6)
  - **module:** Specifies module system (CommonJS) (module.exports, require, etc.)
  - **strict:** True ⟶ Activate use strict
  - **outDir:** (optional) Default route where JS files will be hosted
  - **sourceMap:** True ⟶ Activates source map generation
- **include:** TS files that we want to compile

launch.json configuration (TS)

.vscode > {} launch.json > ...

```jsonc
{
  // Use IntelliSense to learn about possible attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Descriptive Name",
      "program": "${workspaceFolder}/example.js",
      "outFiles": ["${workspaceFolder}/*.js"]
    }
  ]
}
```

launch.json configuration (TS)

- Almost the same as launch.json in JS
- A TS program cannot be debugged directly, it requires prior compilation to convert it into JS. (Reason why format similar)
- Additional configuration:
  - **outFiles:** Help the debugger find all the JS generated from the compilation.

tsc compilation

Process to compile:

- “sudo apt install node-typescript”
- In the directory where tsconfig.json is located or below, execute the **tsc** command

Note: Before debugging, put the breakpoints in the TS file

Repository example

https://github.com/ULL-ESIT-PAI-2023-2024/2023-2024-pai-debugging-stephanBG-taniaVM/tree/master/src/vsc/TypeScript

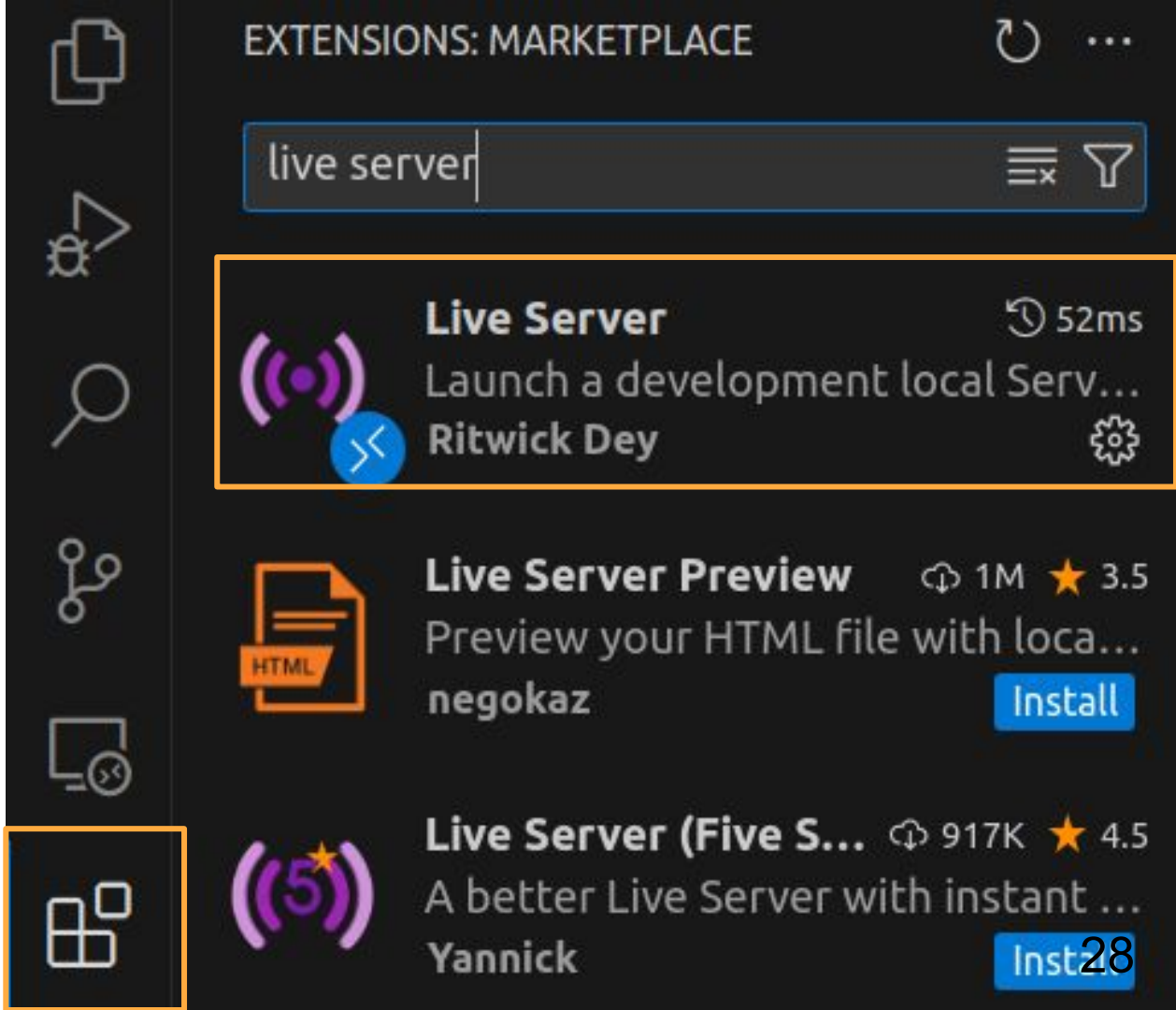
Debugging in the Browser (chrome)

First steps

We will need 4 things:

1. "Live Server" extension in VSC
2. HTML code that calls the function to be debugged
3. On the HTML file, right click. Choose "Open with Live Server" option
4. Once in the browser, F12 to open debugger

1. "Live Server" extension in VSC



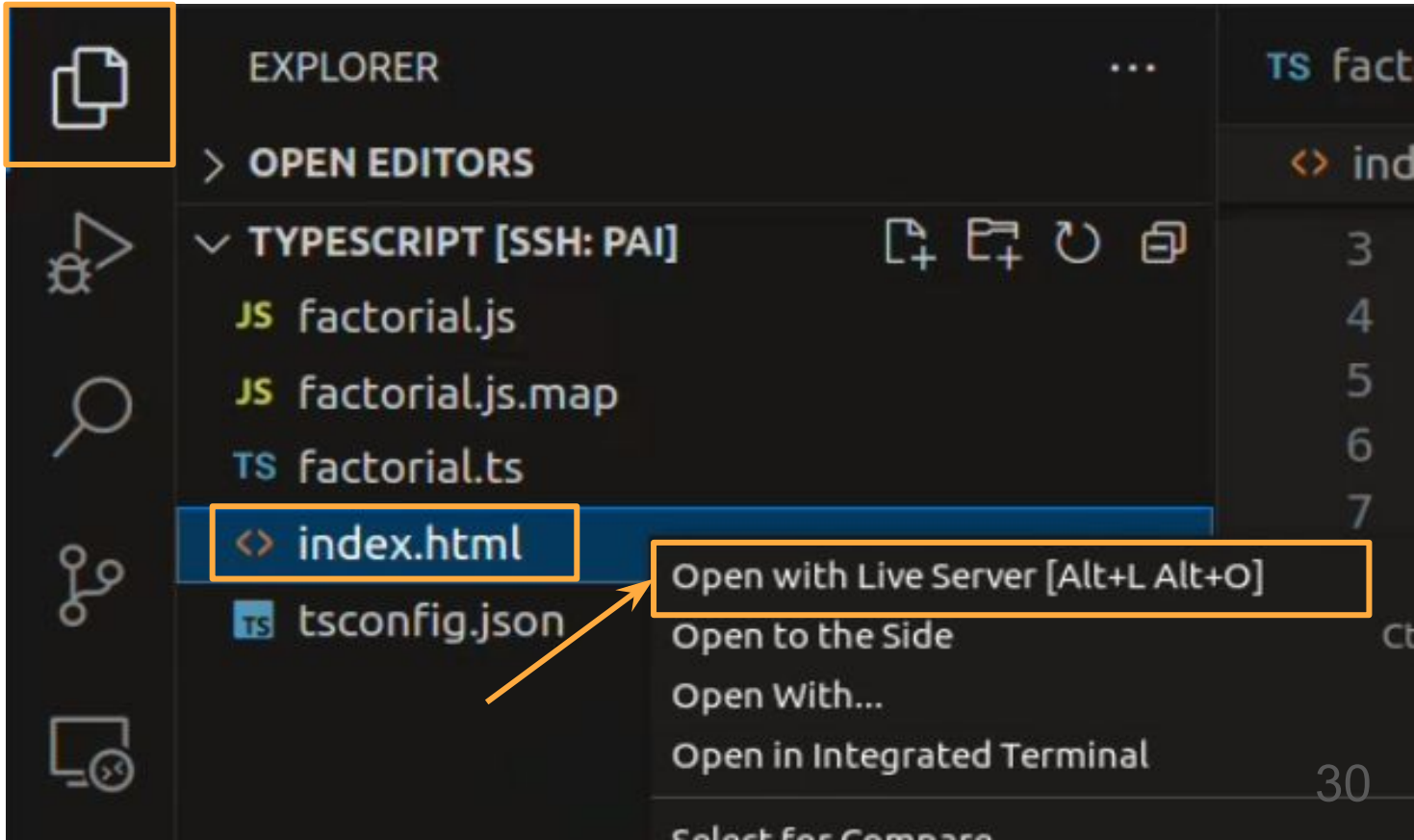
2. HTML code that calls the program to be debugged

If not, the program will not be able to run

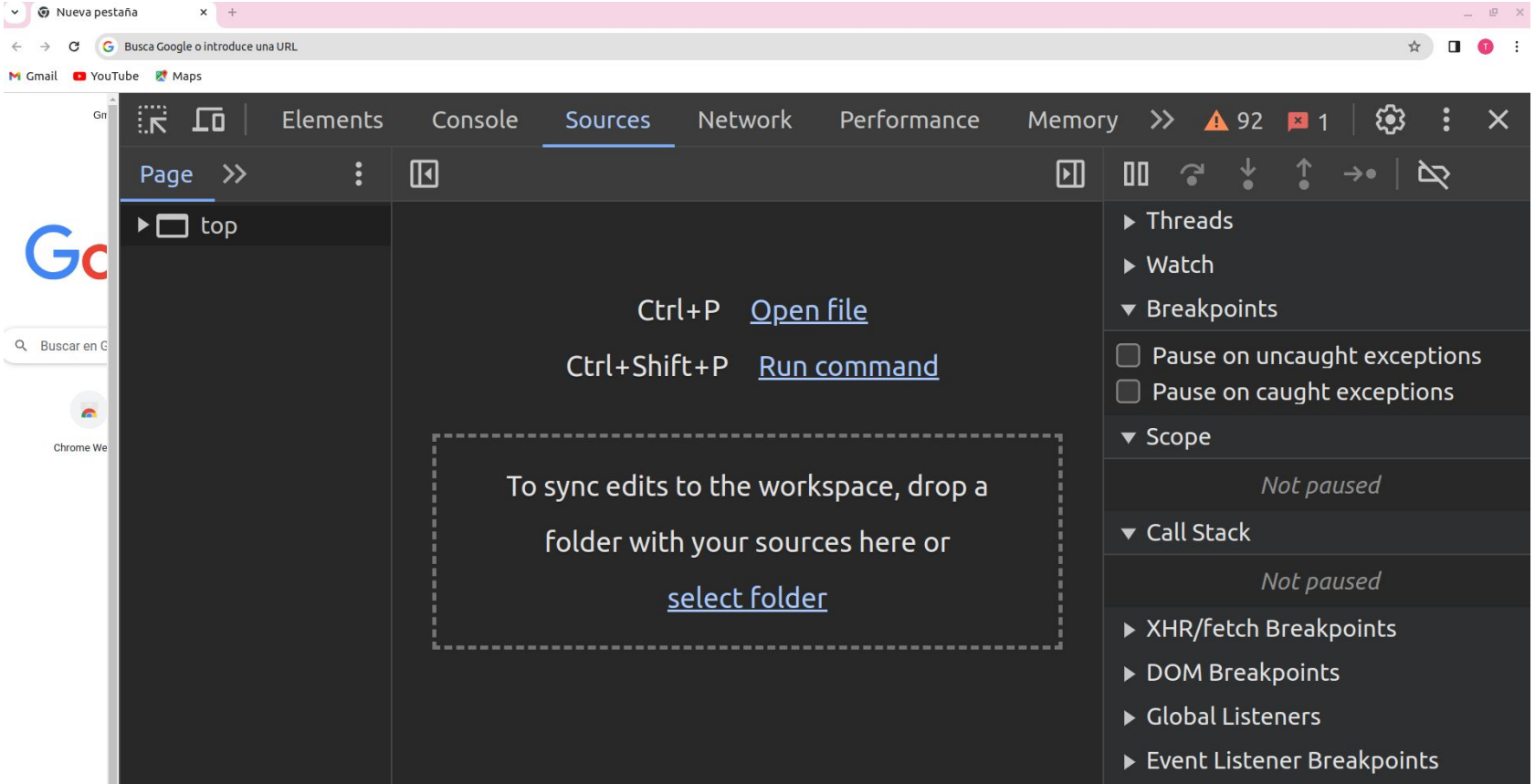
```
<!DOCTYPE html>
<html>
<head>
  <title>factorial</title>
  <script src="factorial.js"></script>
</head>
<body>
  <h2>How to debug TypeScript code?</h2>
  <input type="button" class="button" onclick="startProgram()" value="factorial"/>
</body>
</html>
```

this is necessary to reference the program to be debugged

3. On the HTML file, right click. “Open with Live Server"

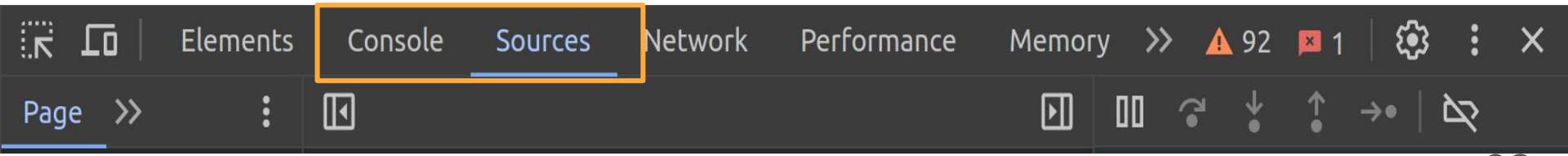


4. Once in the browser, F12 to open debugger



Interface: Top bar

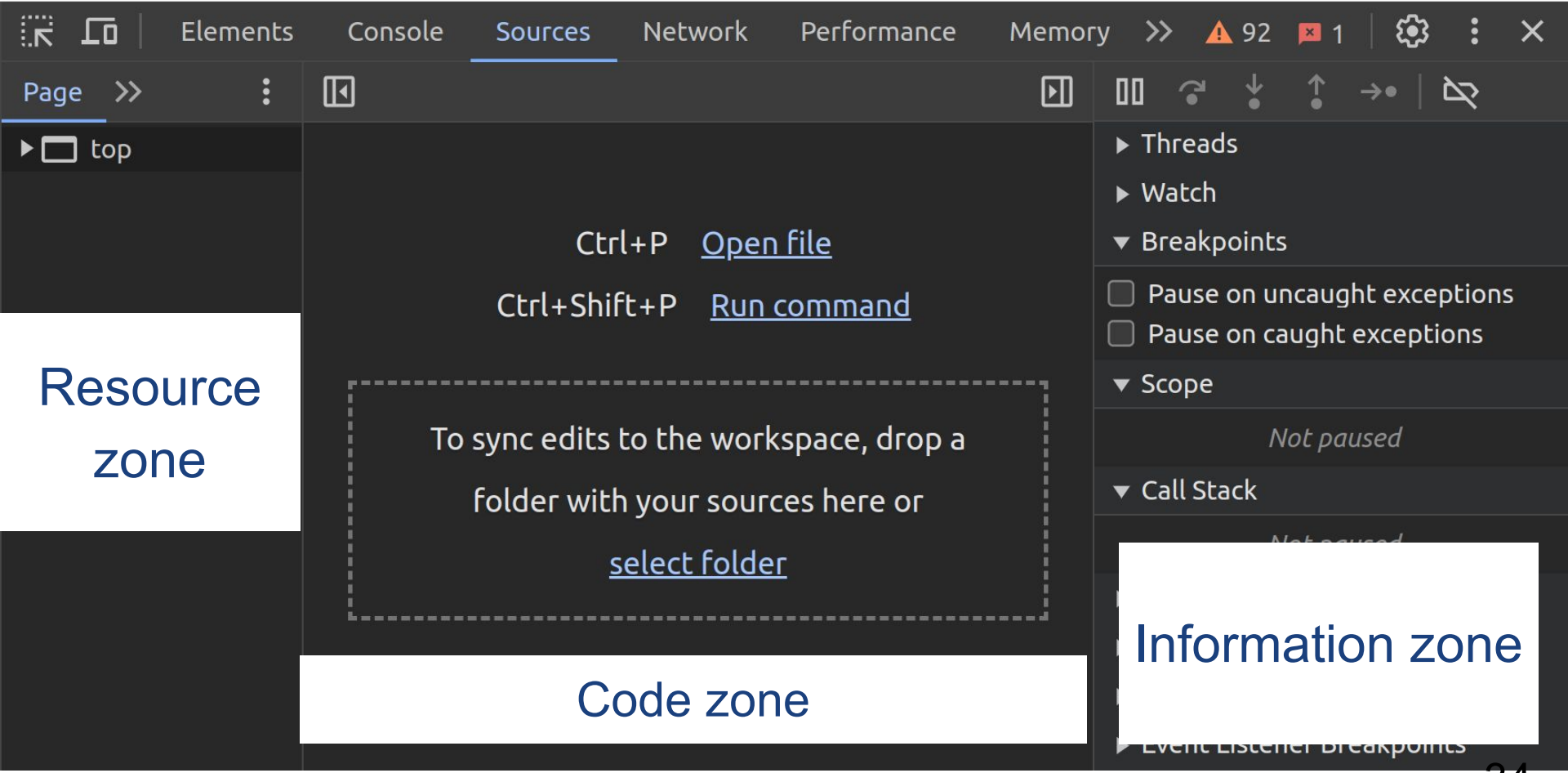
- For debugging, we only need the “console” and "source"
  - Source: Main debugging location
  - Console: to call some program functions individually, test expressions and so on.



Source

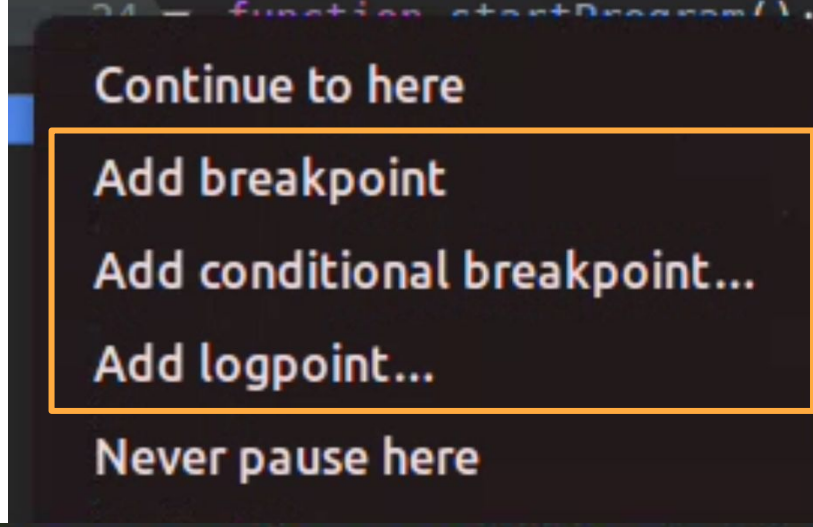
Three areas:

- **Resource zone:** Includes all files
- **Code zone:** Where is all the code
- **Information zone:** Panels to control the debugging process.


Elements
Console
Sources
Network
Performance
Memory
92
1
Page
top
Ctrl+P Open file
Ctrl+Shift+P Run command
To sync edits to the workspace, drop a folder with your sources here or select folder
Threads
Watch
Breakpoints
Pause on uncaught exceptions
Pause on caught exceptions
Scope
Not paused
Call Stack
Resource zone
Code zone
Information zone

Types of breakpoints

- Same breakpoints as in VSC
- debugger command



debugger acts with basic breakpoint

```
function collatzSequence(currentValue) {
  debugger;
  // Stores each number in the sequence
  let collatzSequence = [currentValue];
  while (currentValue !== 1) {
    if (currentValue % 2 === 0) {
      currentValue = currentValue / 2;
    } else {
```

Debugger buttons

Continue until the next point ➡  (before debugging)

Run the following method without inspecting (unless there are breakpoints).

Jump to next command (inspecting)

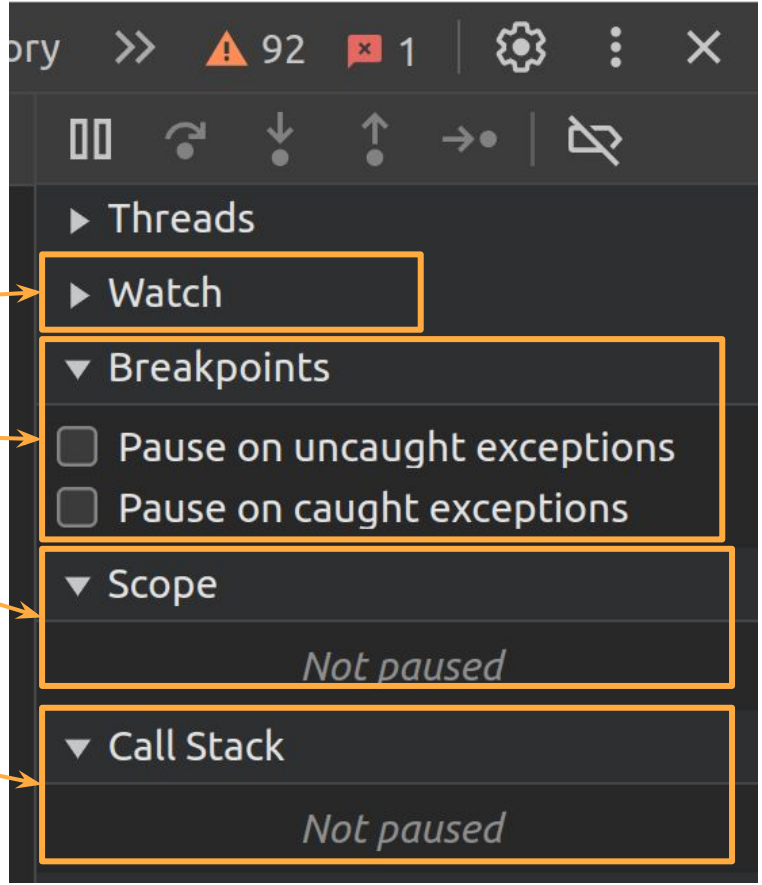
Continue execution until the end (not step by step)

Turn breakpoints on or off

Interface: Panels

The same panels as in VSC

- Watch
- Breakpoints
- Scope (Variables in VSC)
- Call Stack



Debugging in JavaScript (Browser)

Repository example

https://github.com/ULL-ESIT-PAI-2023-2024/2023-2024-pai-debugging-stephanBG-taniaVM/tree/master/src/browser/JavaScript

Debugging in TypeScript (Browser)

Before debugging

1. Create tsconfig.json (as in VSC)
2. Compile with tsc
3. Start debugging as we have seen

Repository example

https://github.com/ULL-ESIT-PAI-2023-2024/2023-2024-pai-debugging-stephanBG-taniaVM/tree/master/src/browser/TypeScript

Bibliography and references

- Debugging in VS Code
- Node.js/JavaScript debugging in VS Code
- Debugging TypeScript
- How to debug Node.js apps in VSC
- Getting started with Node.js debugging in VS Code
- Debugging express application
- Debugging in the browser(ES)