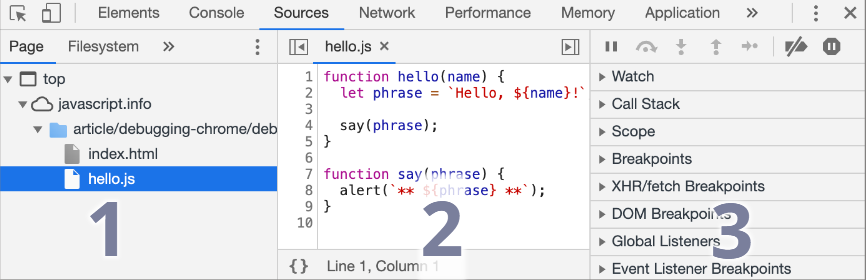
**W04 Reading Notes**

# [**3.1 Debugging in the browser**](https://javascript.info/debugging-chrome)

* Debgugging is the process of finding and fixing errors within a script.
* Broswer tools allow a developer to look at the source code of their site, add breakpoints, andtrace the code step by step which makes debugging much easier.
* **Sources Panel**
  + Section # 1 is the ***File Navigator*** which shows the various files, assets, or extensions.
  + Section # 2 is the ***Code*** ***Editor*** which shows the source code.
  + Section # 2 is for ***JavaScript Debugging*** where you can addbreakpoints and use other debugging tools.

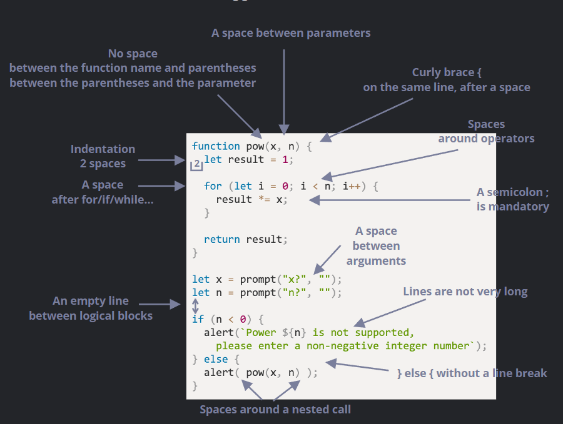
****

* **Console**
  + The { ***Esc*** } key opens the console, then we can type commands, and press the { ***Enter*** } key to execute the command. Its result (whether successful or failure) will appear on the console’s next line(s).
* **Breakpoints**
  + A ***breakpoint*** is a point of code where the debugger automatically pauses the JavaScript executions. This pause allows you to debug the code with the following abilities:
    - Quickly jump to the breakpoint in the code (by clicking on it in the right panel).
    - Temporarily disable the breakpoint by unchecking it.
    - Remove the breakpoint by right-clicking and selecting Remove.
    - …And so on.
  + You can set a ***conditional breakpoint*** that only triggers when the set conditions are met.
  + You can hard code the command { ***debugger;*** } into a JavaScript file and it will be ignored until the browser developmental tools are open. It is like a breakpoint that is premade within the source code rather than targeted separately.
  + **Current Code States:**
    - Watch – shows current values for any expressions
    - Call Stack – shows the chain of nested calls
    - Scope – info about the values of any variables that are currently being used
      * Local variables are items being used that are within the current function.
      * Global variables are items outside the function currently being used.
      * The current value of the { ***this*** } keyword is also shows.
* **Tracing the execution**

|  |  |
| --- | --- |
|  | Resume – this ignores the current breakpoint & resumes the execution. If no other BPs, then the code continues as normal.  Hotkey { ***F8*** } |
|  | Step Over – Run the next command, but don’t go into a function. This is similar to ***Step,*** but if the next statement is a function call, the entire function is executed invisibly rather than stepping through each line of the function.  Hotkey { ***F9*** } |
|  | Step Into –This is similar to ***Step,*** but if the next statement is an async function call, ***Step Into”*** goes into the function code line by line.  Hotkey { ***F11*** } |
|  | Step Out – Continue execution and stop at the last line of the current function. This is helpful when we accidentally enter a function and we need to exit it as soon as possible.  Hotkey { ***Shift+F11*** } |
|  | Step – Run the next command. This will step through all statements one by one. Ignores async functions.  Hotkey { ***F9*** } |
|  | Enable/disable all breakpoints –  This is a mass on/off for all breakpoints |
|  | Enable/disable automatic pause in case of error – When developer tools is open and this command is active, the script will execute everything until an error occurs, then the script automatically pauses so we can see where it dies and what’s the context. |

* ***Continue to here*** allows us to right click on the line of code and execute until that we reach that line without setting a breakpoint.
* **Logging – console.log(***code here***);** allows us to log when certain things occur so we can see what’s going on without opening the debugger.

# [**3.2 Coding Style**](https://javascript.info/coding-style)

****

* Curly braces are used for functions, if/while statements, and a few others.
* Best practice is Egyptian-style to have the **function define** on one line with an opening curly brace, then the function code block on separate lower indented lines, then the closing curly brace on its own line.
* When writing large blocks of code in backtick quotes, if statements, or functions: long horizontal lines are not easily readable so it’s best to split them to new lines where it makes sense.
* Idents:
  + Code inside a function should be horizontally indented a few spaces to show what function the code belongs to.
  + Empty lines should be placed to split the code into logical blocks.
* Semicolons should be present after each statement.
* It’s best to avoid deep nesting so if a function/if statement can be readable on only one line, do that instead of nesting.
* There are two main ways to declare functions: 1) declare the function by itself, then attribute that function to a variable. 2) declare the variable and immediately declare the function inside the variable.
* You can also declare the entire function where it is called, but that is not recommended.
* These companies/groups have style guides that recommend how to style JavaScript: Google, Airbnb, Idiomatic.JS, StandardJS.
* Linters are tools that auto-check your code and suggest improvements and bug fixes.
* Common Linters – JSLint, JSHint, ESLint
* Many IDEs have built in linting.

# [**10.1 Error handling, "try...catch"**](https://javascript.info/try-catch)

* ***try…catch*** is used to “try” a code block & if everything works, then the catch block is ignored and the rest of the code passes as normal. If an error occurs, the catch block runs its code.
* ***try…catch*** only works for runtime errors (aka ***exceptions***) which means the code needs to be runnable & valid JavaScript. If the code is syntactically wrong, then the ***try…catch*** will not work.
* ***try…catch*** works synchronously so it won’t catch any errors that are in async scheduled code, such as ***setTimeout***. To use ***try…catch*** to read anything in a scheduled function like ***setTimeout,*** you must make a new ***try…catch*** block inside that scheduled function.
* Users don’t like when code dies w/ no explanation so ***try…catch*** is used to catch the code dying and throw an alert/message and maybe retry the attempt.
* Sometimes the code can be syntactically correct & won’t throw any errors. You can use “***throw***” operator to get around that by forcing an error to be thrown by running one of the constructor’s below and pass in a string. The constructor will return that string as a ***message*** and will jump into ***catch*** and run the ***catch***’scode.
* Common constructors for standard errors are ***Error, SyntaxError, ReferenceError, TypeError,*** and others.
* The ***catch*** code block runs the same code no matter what error occurs so we can avoid that by checking the error’s type with an if statement & alert a type-specific message.
* The operator ***finally*** is a code clause that is optional and if it exists, it always runs no matter if there were any errors or not. That means even if ***try*** runs w/ no errors, or ***catch*** runs with errors, then when either is finished, ***finally*** will run its code.
* ***try & finally*** can both be used together without using catch if we want to run the code & ensure the process finishes, even if we do not want to catch any errors.