Welcome to the CoGrammar JavaScript Lecture

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Skills Bootcamp 8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar

JavaScript

March 2024

Lecture Overview

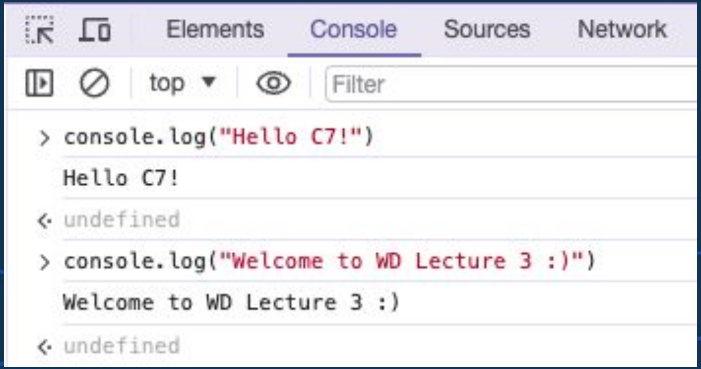
→ Variables
→ Data Types
→ Conditional Statements
→ Trace Tables
→ Stack Traces
→ Debugging

CoGrammar

JavaScript

A versatile scripting language utilised in front-end web development and server-side programming.

- ❖ We use JavaScript with HTML and CSS to transform our **static web pages** to **dynamic web pages.**
- ❖ Last week, we learnt how to **link scripts** to our HTML. These scripts are written in **JavaScript**.
- ❖ Browsers have **built-in consoles** used to **debug** JavaScript code.



```
> console.log("Hello C7!")
  Hello C7!
<· undefined
> console.log("Welcome to WD Lecture 3 :)")
  Welcome to WD Lecture 3 :)
<· undefined
```

CoGrammar

JavaScript

- The console is useful for **debugging** and running **code snippets**.
- To create our scripts, we will use **Visual Studio Code** and **Node.js**.
- The following extensions are also helpful when running your code:
  - Code Runner: allows you to run JavaScript code in VS Code by pressing **Ctrl+Alt+N**, right-clicking and pressing **“Run Code”**, or by pressing the **“Play” button**.
  - Open in Browser: allows you to open an HTML file which has been correctly linked to JavaScript scripts **in your default browser**.

CoGrammar

Variables

Symbols used to represent values stored in the computer's memory

❖ The special word (keyword) **let** indicates that this program is going to **define a variable**.

❖ It is followed by the **name** of the variable, **“=” operator** and a **value/expression**.

```
let num1 = 5;
let sum = 5+5;
```

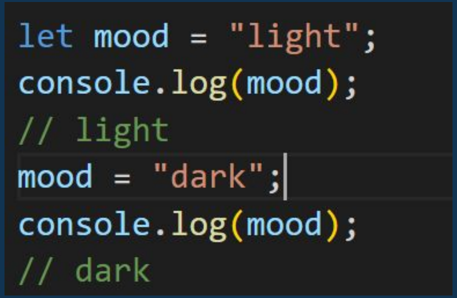
❖ After a variable has been defined, its **name** can be used in expressions.

```
console.log("Your number is: ")
console.log(num1)
```

CoGrammar

Variables

- When a value is bound to a variable, it **does not** mean the value bound to the variable **cannot change.**
- The **“=” operator** can be used at any time on **existing variables** to **reassign** a new value to that variable.

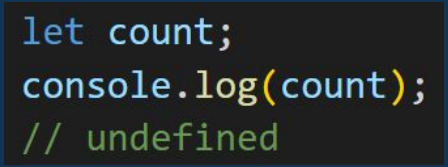


```
let mood = "light";
console.log(mood);
// light
mood = "dark";
console.log(mood);
// dark
```

CoGrammar

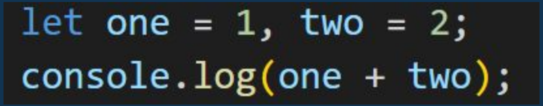
Variables

- ❖ If you ask for the **value** of an empty binding, you'll get the value **undefined**.



```
let count;
console.log(count);
// undefined
```

- ❖ A **single let** statement may define **multiple bindings**. The definitions must be separated by commas.

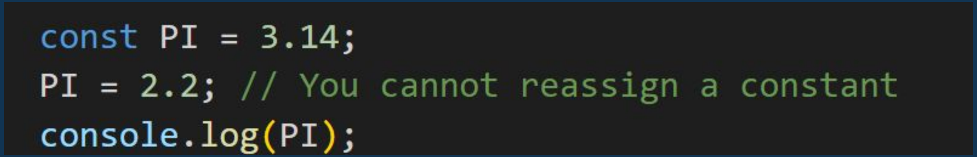


```
let one = 1, two = 2;
console.log(one + two);
```

CoGrammar

Variables

- The keyword **const** can also be used to create variables, in a similar fashion to **let.**
- The keyword **const** stands for **constant**.
- It defines a **constant variable**, which stores the **same value** for as long as it exists.

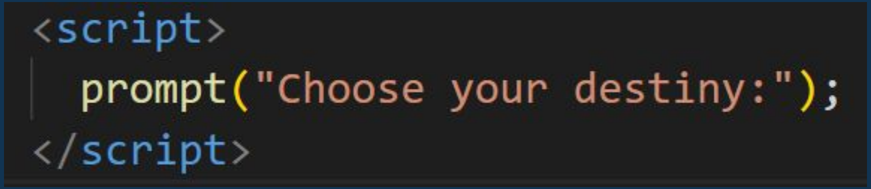


```
const PI = 3.14;
PI = 2.2; // You cannot reassign a constant
console.log(PI);
```

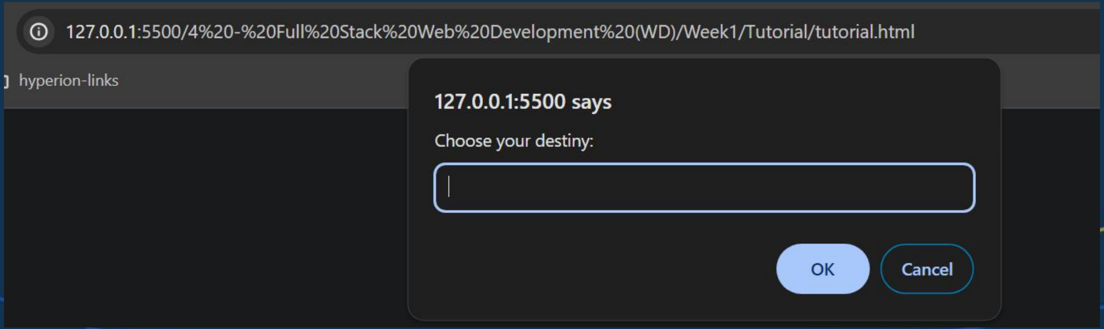
CoGrammar

Functions: prompt

- **prompt**: a function that shows a little dialog box asking for user input.



```
<script>
  prompt("Choose your destiny:");
</script>
```



```
127.0.0.1:5500/4%20-%20Full%20Stack%20Web%20Development%20(WD)/Week1/Tutorial/tutorial.html
```

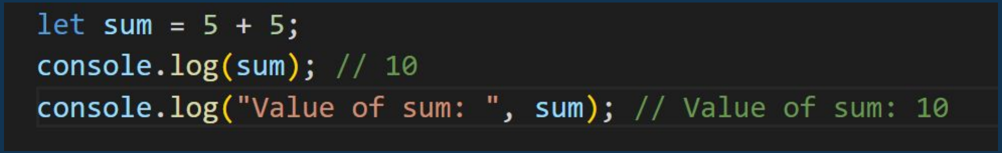
127.0.0.1:5500 says

Choose your destiny:

CoGrammar

Functions: console.log

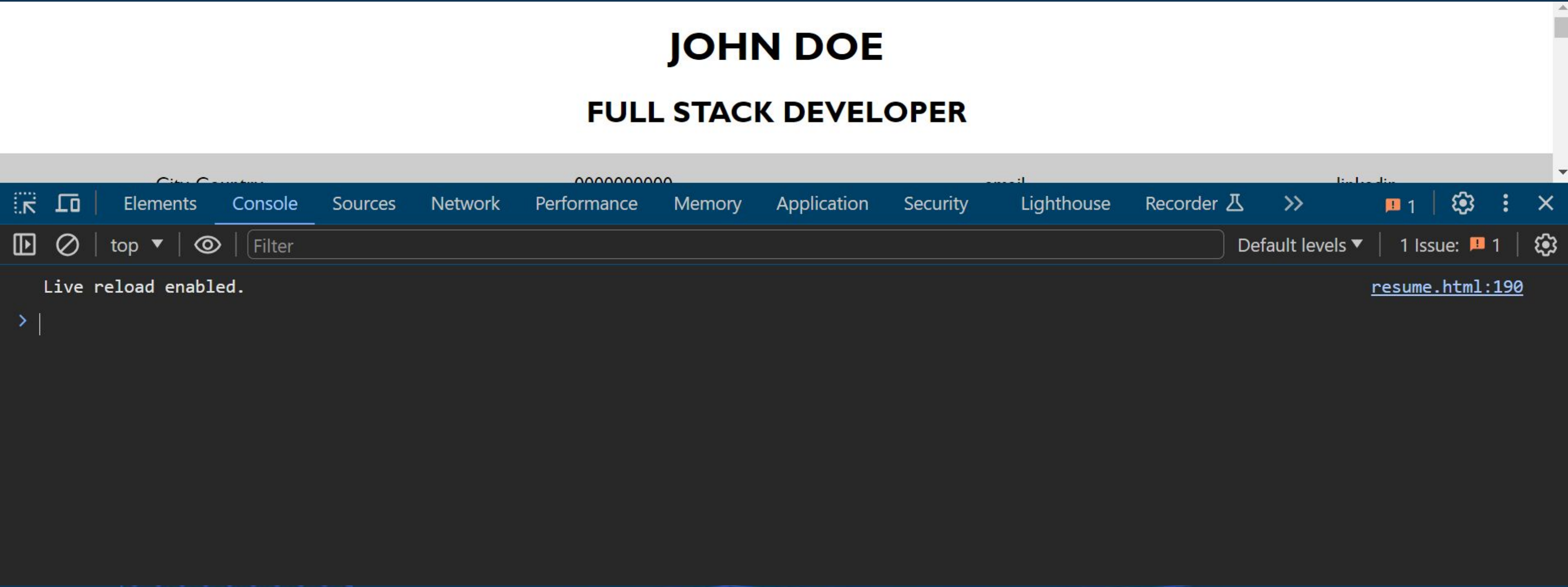
- Most JavaScript systems (including all modern web browsers and Node.js) provide a **console.log** function that writes out its **arguments** to some text **output** device.
- In browsers, the output lands in the **JavaScript console**. This part of the browser interface is hidden by default, but most browsers open it when you open **Developer Tools or the Inspect view**.



```
let sum = 5 + 5;
console.log(sum); // 10
console.log("Value of sum: ", sum); // Value of sum: 10
```

CoGrammar

Functions: console.log



JOHN DOE

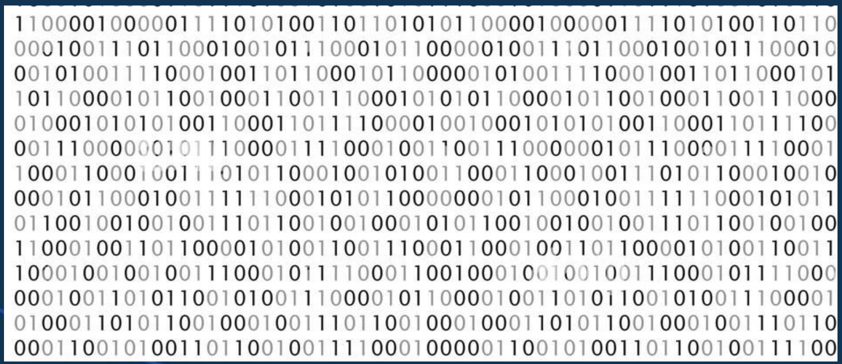
FULL STACK DEVELOPER

```
Live reload enabled.                                                    resume.html:190
>
```

CoGrammar

Data Types

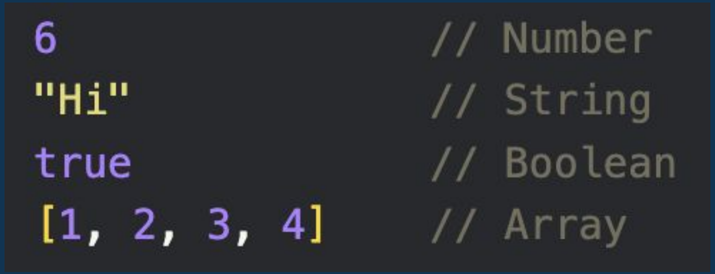
- A typical modern computer has more than **100 billion bits** in its volatile data storage (**working memory**).
- To be able to work with such quantities of **bits** without getting lost, we separate them into **chunks** that represent **pieces of information**.



CoGrammar

Data Types

- In JavaScript, those chunks are called **values**.
- Every **value** has a **type** that determines its **role**.
- Some values are numbers, some values are pieces of text and so on.



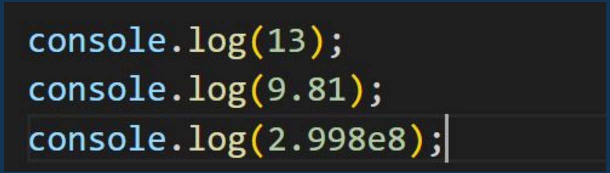
```
6                   // Number
"Hi"                // String
true                // Boolean
[1, 2, 3, 4]        // Array
```

CoGrammar

Numbers

Values of the number type are numeric values.

- **Fractional numbers** are written using a dot.
- For very big or very small numbers, you may also use **scientific notation** by adding an **e (for exponent)**.

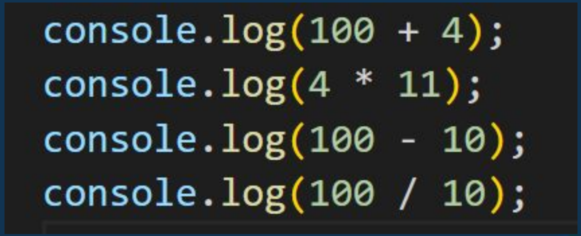


```
console.log(13);
console.log(9.81);
console.log(2.998e8);
```

CoGrammar

Arithmetic Operations

- The **+** and ***** symbols are called **operators**.
- **Operators** are used to represent **operations**, the former being **addition** and the latter being **multiplication**.
- Putting an operator between two values will **apply** it to those values and produce a **new value**.
- We use **-** for **subtraction** and **/** for division.

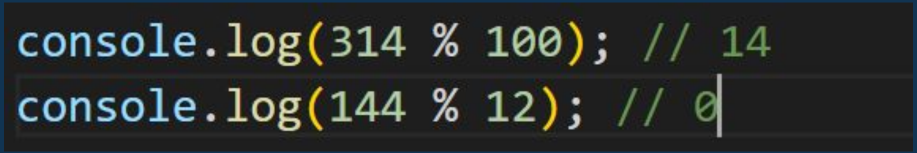


```
console.log(100 + 4);
console.log(4 * 11);
console.log(100 - 10);
console.log(100 / 10);
```

CoGrammar

Arithmetic Operations

❖ The **%** symbol is used to represent the **remainder** operation. You’ll also often see this operator referred to as **modulo**.



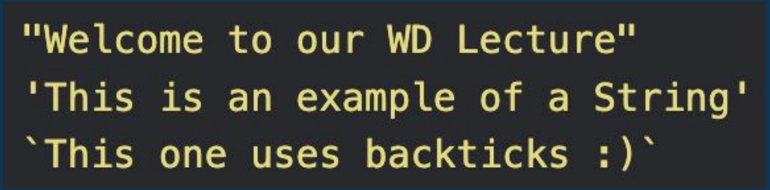
```
console.log(314 % 100); // 14
console.log(144 % 12); // 0
```

CoGrammar

Strings

Strings are used to represent text.

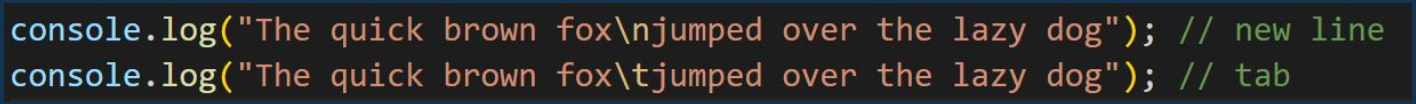
- They are written by enclosing their content in **quotes**.
- You can use **single quotes**, **double quotes**, or **backticks** to mark strings, as long as the quotes at the start and the end of the string match.



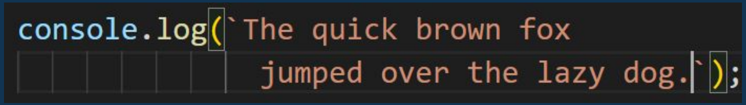
```
"Welcome to our WD Lecture"
'This is an example of a String'
`This one uses backticks :)`
```

Strings

- A **backslash (\\)** inside quoted text indicates that the character after it has a special meaning. This is called **escaping** the character.
- Newlines can be included only when the string is quoted with backticks (`).



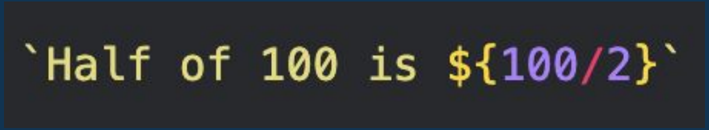
```
console.log("The quick brown fox\njumped over the lazy dog"); // new line
console.log("The quick brown fox\tjumped over the lazy dog"); // tab
```



```
console.log(`The quick brown fox
             jumped over the lazy dog.`);
```

Template literals

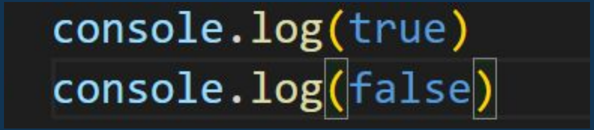
- Backtick-quoted strings, usually called **template literals**, can do a few more tricks.
- Apart from being able to span lines, they can also **embed other values**.
- When you write something inside **${}** in a **template literal**, its result will be **computed**, **converted to a string**, and included at that position.



```
`Half of 100 is ${100/2}`
```

Boolean values

- It is often useful to have a value that distinguishes between only two possibilities, like “yes” and “no” or “on” and “off”.
- For this purpose, JavaScript has a **Boolean** type, which has just two values, **true** and **false**, written as those words.

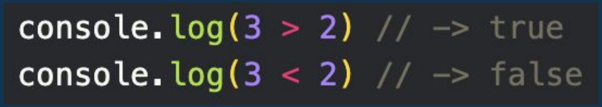


```
console.log(true)
console.log(false)
```

CoGrammar

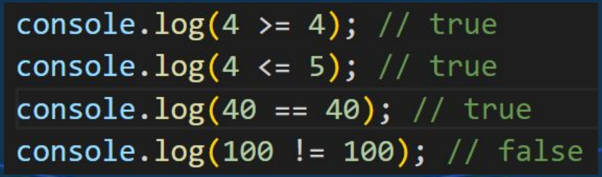
Comparison Operations

- The **>** and **<** signs are the traditional symbols for **"is greater than"** and **"is less than"**, respectively.
- Applying them results in a Boolean value that indicates whether they hold true in this case.



```
console.log(3 > 2) // -> true
console.log(3 < 2) // -> false
```

- Other similar operators are **>= (greater than or equal to)**, **<= (less than or equal to)**, **== (equal to)**, and **!= (not equal to)**.

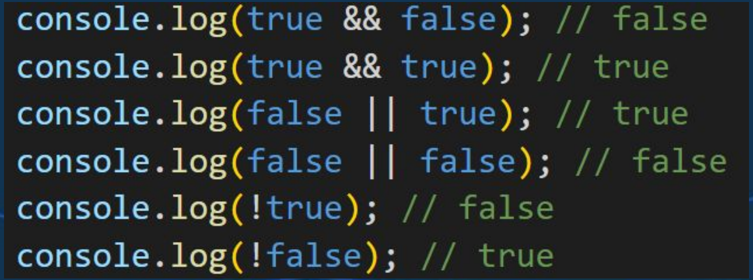


```
console.log(4 >= 4); // true
console.log(4 <= 5); // true
console.log(40 == 40); // true
console.log(100 != 100); // false
```

CoGrammar

Logical Operators

- ❖ JavaScript supports three logical operators: **&&**, **||**, and **!**.
- ❖ The **&&** operator represents logical **AND**
  - ➢ Its result is **true** only if **both** the values given to it are **true**.
- ❖ The **||** operator denotes logical **OR**.
  - ➢ Its result is **true** if **either** the values given to it are **true**.
- ❖ **Not** is written as an **exclamation mark (!)** and it flips the value given to it.
  - ➢ **!true** produces **false** and **!false** gives **true**.



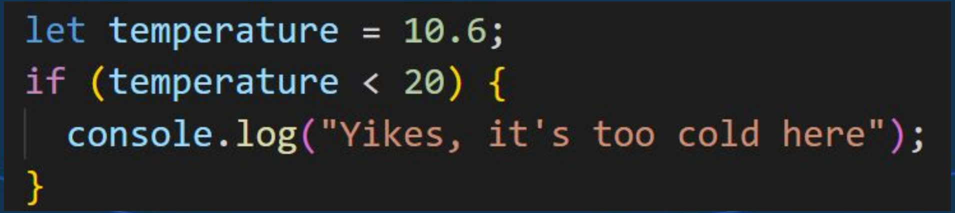
```
console.log(true && false); // false
console.log(true && true); // true
console.log(false || true); // true
console.log(false || false); // false
console.log(!true); // false
console.log(!false); // true
```

CoGrammar

Conditional Statements

Statements that perform different actions depending on whether a condition evaluates to true or false.

- **Conditional execution** is created with the **if** keyword in JavaScript.
- We want some code to be executed **if**, and only **if**, a certain **condition** holds.
- The deciding expression is written after the **if** keyword, between parentheses, followed by the statement to execute.

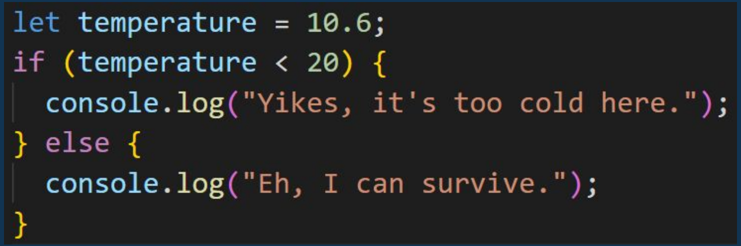


```javascript
let temperature = 10.6;
if (temperature < 20) {
  console.log("Yikes, it's too cold here");
}
```

CoGrammar

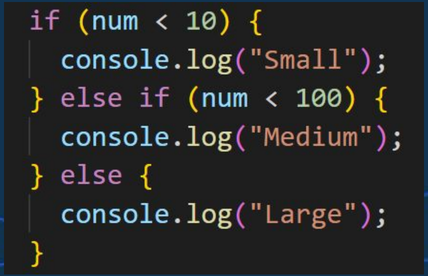
Conditional Statements

- You can use the **else keyword**, together with **if**, to create two separate, **alternative execution** paths.



```
let temperature = 10.6;
if (temperature < 20) {
  console.log("Yikes, it's too cold here.");
} else {
  console.log("Eh, I can survive.");
}
```

- If you have more than two paths to choose from, you can “chain” multiple if/else pairs together.



```
if (num < 10) {
  console.log("Small");
} else if (num < 100) {
  console.log("Medium");
} else {
  console.log("Large");
}
```

CoGrammar

Comments

A piece of text that is part of a program but is ignored by the computer.

❖ You might just want to include some related thoughts as part of your program. This is what **comments** are for.

Single line comments

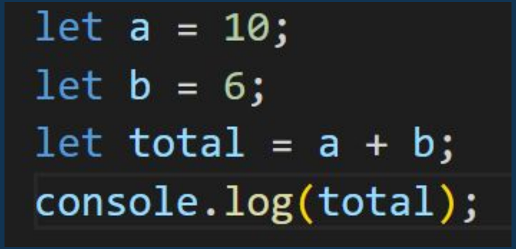
```
// It's a green hollow where a river sings
```

Multiline comments

```
/*
  I first found this number scrawled on the back of an old
  notebook. Since then, it has often dropped by, showing up in
  phone numbers and the serial numbers of products that I've
  bought. It obviously likes me, so I've decided to keep it.
*/
```

Trace Tables

A technique used to test a program and predict, step-by-step, how the computer will run it.



```
let a = 10;
let b = 6;
let total = a + b;
console.log(total);
```

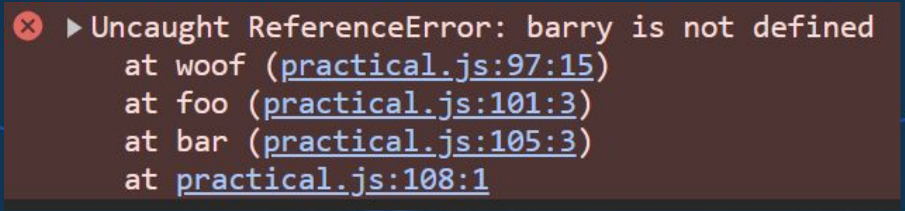
line	a	b	total	log
1	10			
2		6		
3			16	
4				16

CoGrammar

Stack Trace

A detailed report of function calls leading to an error.

- Analyzing the stack trace helps pinpoint the **exact location of the error**.
- **Woof, foo and bar** are the functions that were called.
  - The bottom most line shows the line number where **bar** was called, it says the function was called at line **108**.
  - From here we can see that **bar** was called first, which later called the function **foo.**
  - After that **foo** called the function **woof.** This indicates that the source of the error is **woof.**



```
▶Uncaught ReferenceError: barry is not defined
    at woof (practical.js:97:15)
    at foo (practical.js:101:3)
    at bar (practical.js:105:3)
    at practical.js:108:1
```

CoGrammar

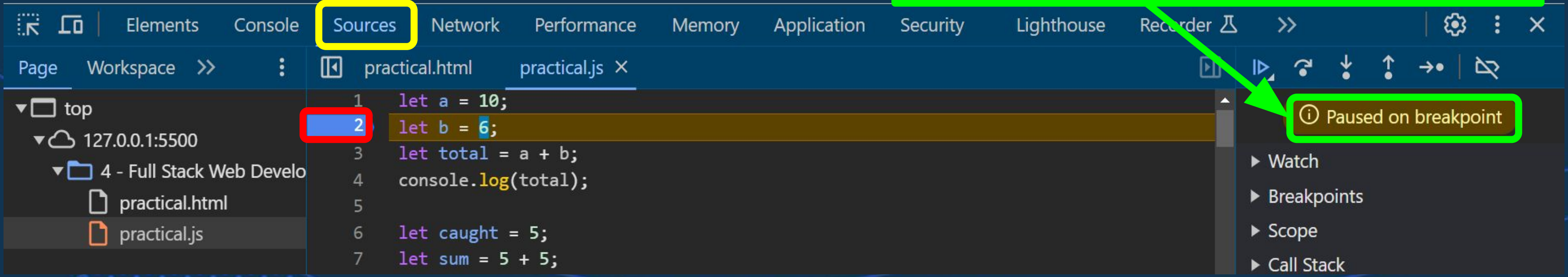
Debugging JavaScript

The process of examining the program, finding the error and fixing it.

❖ You can set **breakpoints** for JavaScript code in the **Sources** tab in the **Developers tool**.

❖ JavaScript will stop executing at each **breakpoint** and lets you examine the values.

These buttons help you move around the code in debug mode.



```
let a = 10;
let b = 6;
let total = a + b;
console.log(total);

let caught = 5;
let sum = 5 + 5;
```

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar