Welcome to the CoGrammar Collections and Iterations 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

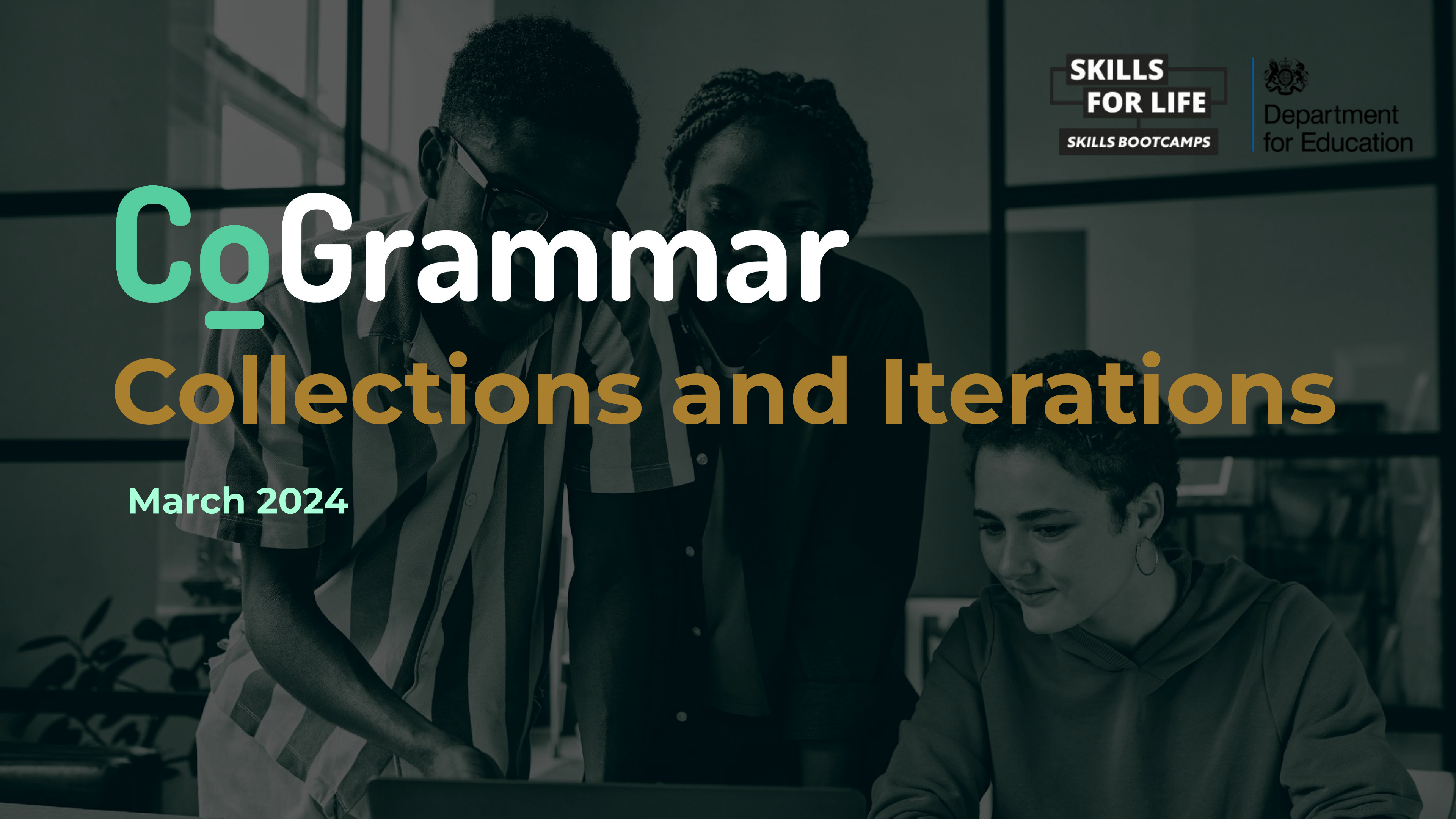
✅ Criterion 3: Course Progress

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

Collections and Iterations

March 2024

Lecture Overview

- → While Loops
- → For Loops
- → Strings
- → Arrays
- → Maps

CoGrammar

Loops

CoGrammar

Loops

- Consider a program that outputs all even numbers from 0 to 12. One way to write this is as follows:

```
console.log(0);
console.log(2);
console.log(4);
console.log(6);
```

And so on……

CoGrammar

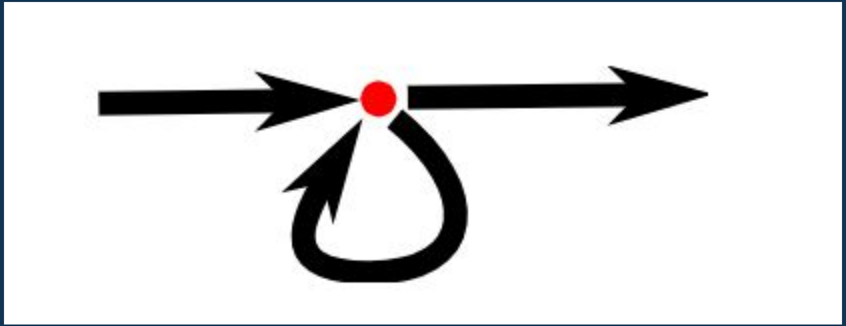
Loops

- That works, but the idea of writing a program is to make something **less work**, not more.
- If we needed all even numbers **less than 1,000**, this approach would be **unworkable**.
- What we need is a way to run a piece of code multiple times. This form of control flow is called a **loop**.

CoGrammar

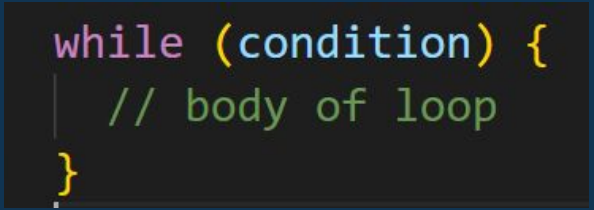
Loops

❖ **Looping** control flow allows us to go back to some point in the program where we were before and **repeat** it.



While Loops

- The screenshot below shows the syntax of **while** loops.

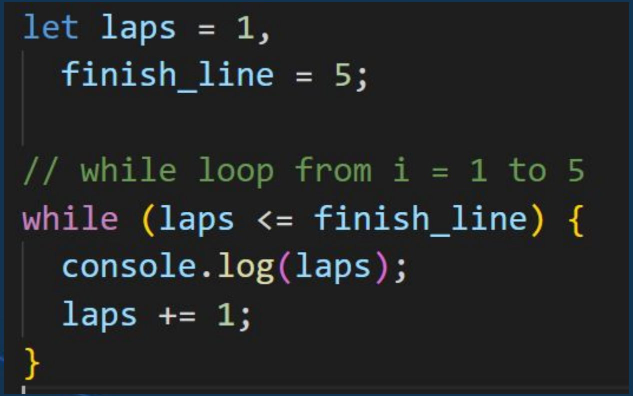


```
while (condition) {
  // body of loop
}
```

- **While** loops are used when you need to repeat your code until a certain **condition is met**.
- We can use **trace tables** to help us test our loops and evaluate how the computer will run the code, line by line.

CoGrammar

While Loops



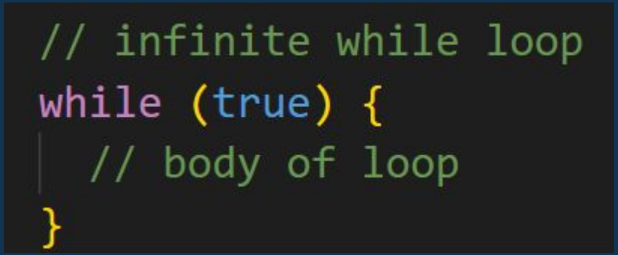
```
let laps = 1,
  finish_line = 5;

// while loop from i = 1 to 5
while (laps <= finish_line) {
  console.log(laps);
  laps += 1;
}
```

laps	finish_line	laps <= finish_line	Output
1	5	true	Print 1
2	5	true	Print 2
3	5	true	Print 3
4	5	true	Print 4
5	5	true	Print 5
6	5	false	Stop

CoGrammar

Infinite While Loops

❖ If the **condition** of a **loop** is always true, the **loop** runs for **infinite** times.

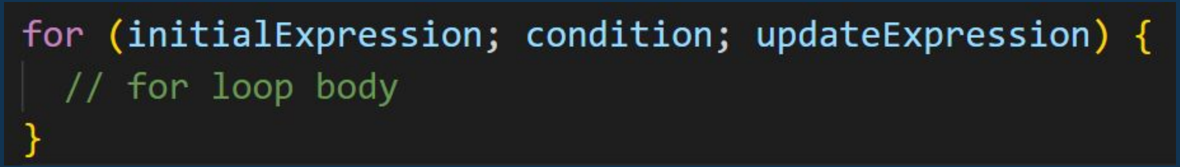


```
// infinite while loop
while (true) {
  // body of loop
}
```

CoGrammar

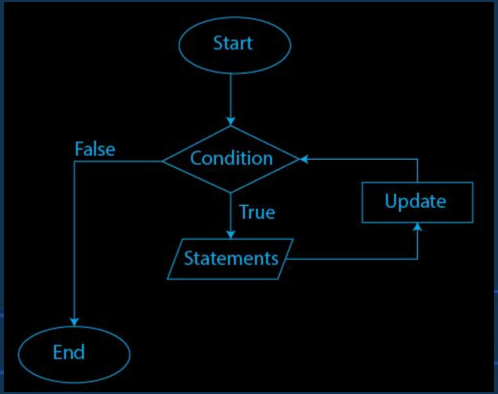
For Loops

❖ The screenshot below shows the syntax of **for** loops.



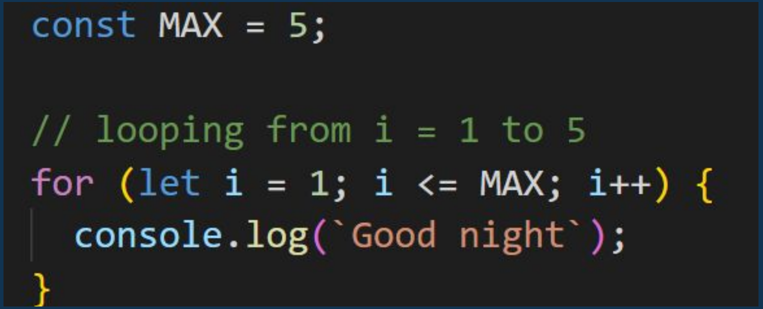
```
for (initialExpression; condition; updateExpression) {
  // for loop body
}
```

❖ **For** loops are used when we need to repeat our code a **set number of times**.



CoGrammar

For Loops



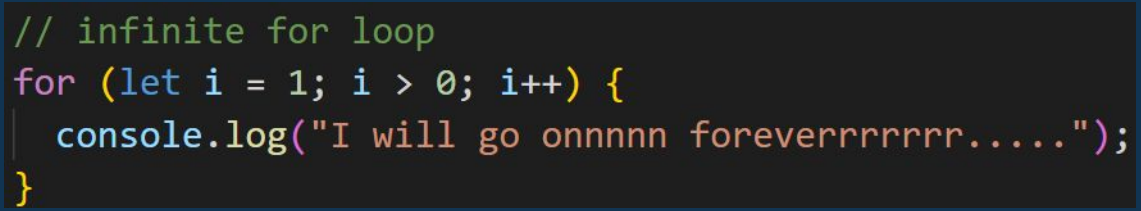
```
const MAX = 5;

// looping from i = 1 to 5
for (let i = 1; i <= MAX; i++) {
  console.log(`Good night`);
}
```

i	MAX	i <= MAX	Action
1	5	true	Print
2	5	true	Print
3	5	true	Print
4	5	true	Print
5	5	true	Print
6	5	false	Stop

CoGrammar

Infinite For loop

❖ If the test **condition** in a **for** loop is always **true**, it runs forever (until memory is full).



```
// infinite for loop
for (let i = 1; i > 0; i++) {
  console.log("I will go onnnnn foreverrrrrrr.....");
}
```

CoGrammar

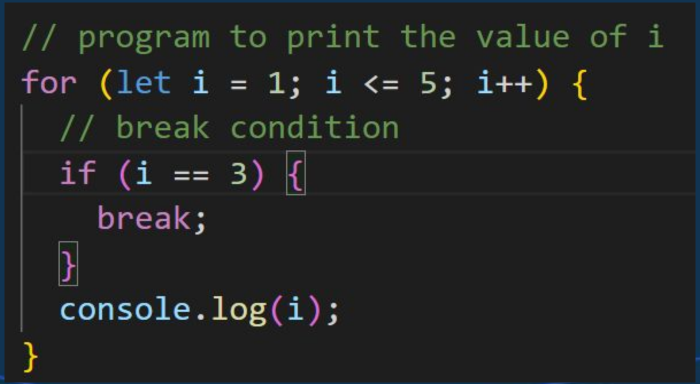
For vs While

- A **for loop** is usually used when the number of iterations is **known**.
- The **while loop** is usually used when the number of iterations is **unknown.**

CoGrammar

Break Statement

- The **break** statement is used to **terminate** the loop immediately when it is encountered.
- You can run a **break statement** by using the **break** keyword.
- This works for both **while** and **for** loops.

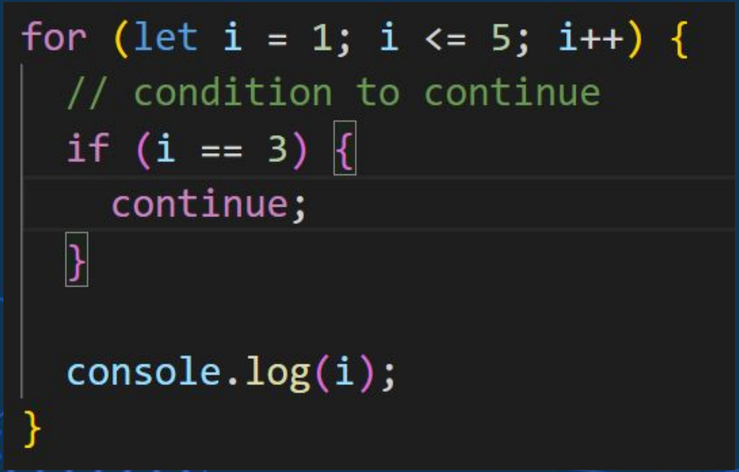


```
// program to print the value of i
for (let i = 1; i <= 5; i++) {
  // break condition
  if (i == 3) {
    break;
  }
  console.log(i);
}
```

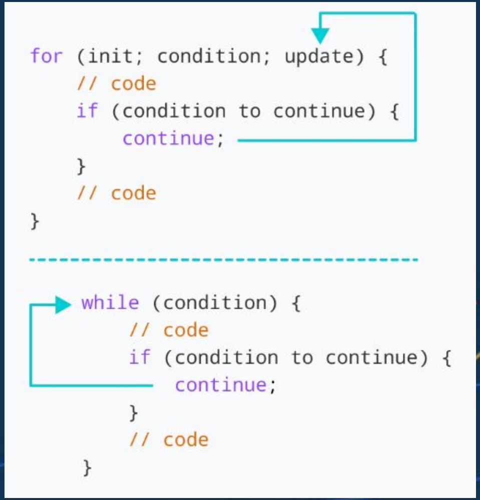
CoGrammar

Continue Statement

- The **continue** statement is used to **skip** the current iteration of the loop and the control flow of the program goes to the **next iteration**.
- This works for both **while** and **for** loops.



```
for (let i = 1; i <= 5; i++) {
  // condition to continue
  if (i == 3) {
    continue;
  }

  console.log(i);
}
```



```
for (init; condition; update) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}
```

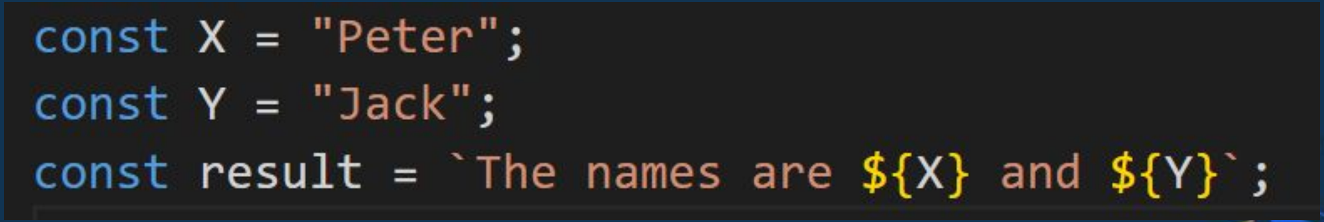
```
while (condition) {
    // code
    if (condition to continue) {
        continue;
    }
    // code
}
```

CoGrammar

Strings

Strings

- JavaScript **string** is a primitive data type that is used to work with **texts**.
- In JavaScript, strings are created by surrounding them with quotes.



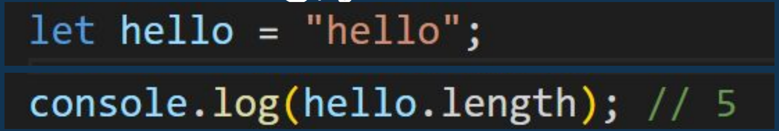
```
const X = "Peter";
const Y = "Jack";
const result = `The names are ${X} and ${Y}`;
```

CoGrammar

Strings

- You can access string characters using their position.

- To find the length of a string, you can use built-in **length** property.



```
let hello = "hello";

console.log(hello.length); // 5
```

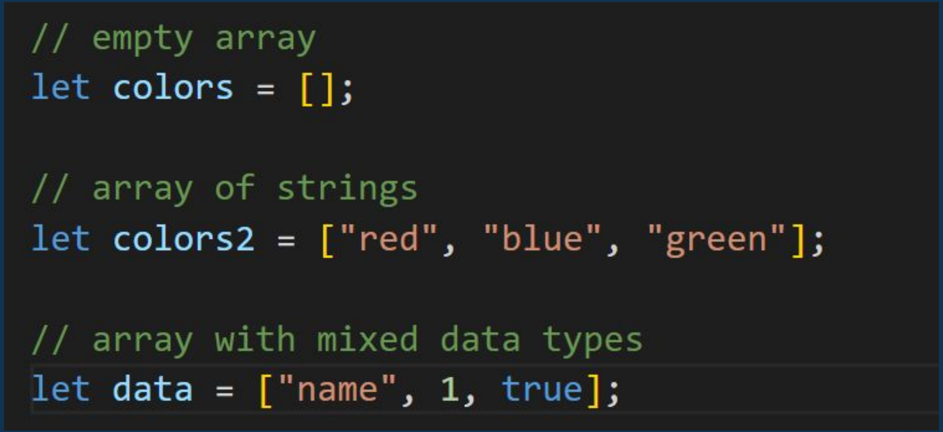
CoGrammar

Arrays

Arrays

An array is a data structure that can store multiple values at once.

- ❖ We can create an **array** by placing elements inside an array literal **[ ]**, separated by **commas.**

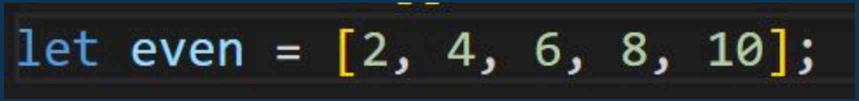


```
// empty array
let colors = [];

// array of strings
let colors2 = ["red", "blue", "green"];

// array with mixed data types
let data = ["name", 1, true];
```

CoGrammar

Arrays

- ❖ Each **element** of an array is associated with a number called an **index**.
- ❖ The **index** specifies the **position** of the element inside the array.

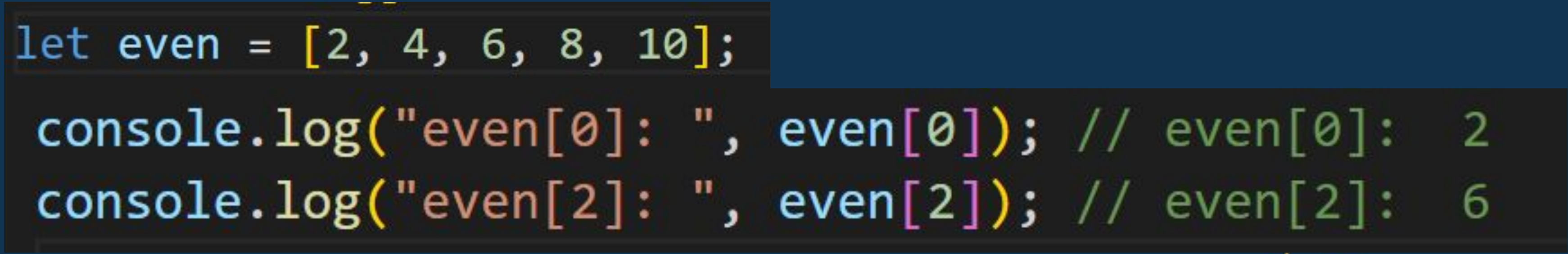


```
let even = [2, 4, 6, 8, 10];
```

Index	0	1	2	3	4
	2	4	6	8	10

Arrays

❖ We can use an array **index** to **access** the elements of the array.

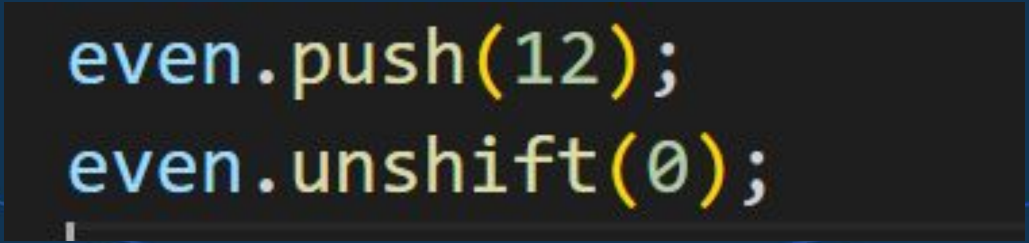


```
let even = [2, 4, 6, 8, 10];
console.log("even[0]: ", even[0]); // even[0]:  2
console.log("even[2]: ", even[2]); // even[2]:  6
```

CoGrammar

Arrays

- We can **add** elements to an array using **built-in** methods like **push()** and **unshift()**.
  - The push() method adds an element at the **end** of the array.
  - The unshift() method adds an element at the **beginning** of the array.

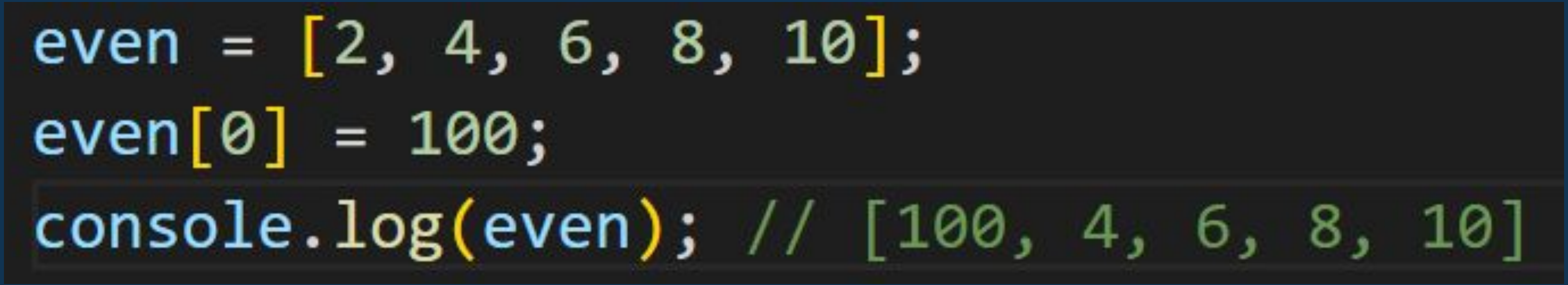


```
even.push(12);
even.unshift(0);
```

CoGrammar

Arrays

- We can add or **change** elements by accessing the **index** value.

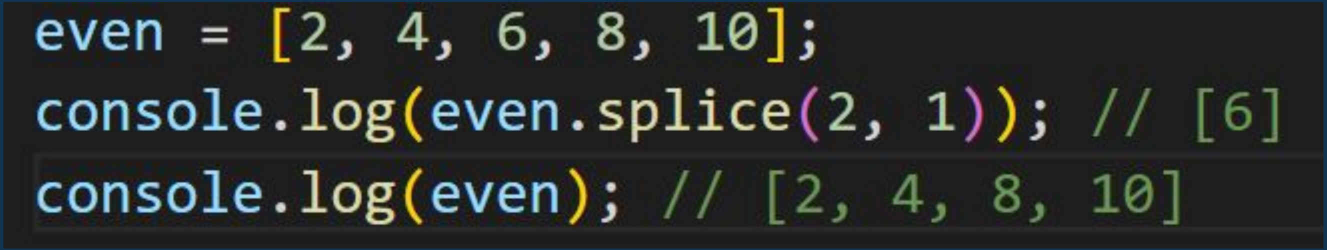


```
even = [2, 4, 6, 8, 10];
even[0] = 100;
console.log(even); // [100, 4, 6, 8, 10]
```

CoGrammar

Arrays

❖ We can remove an element from any specified index of an array using the **splice()** method.

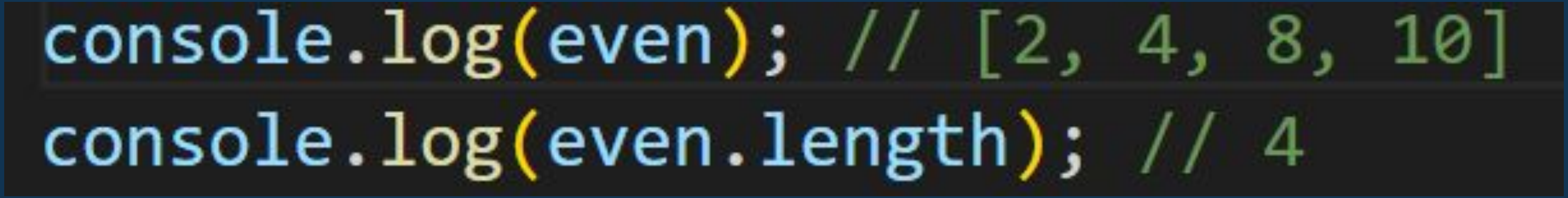


```
even = [2, 4, 6, 8, 10];
console.log(even.splice(2, 1)); // [6]
console.log(even); // [2, 4, 8, 10]
```

CoGrammar

Arrays

❖ We can find the length of an array using the **length** property.



```
console.log(even); // [2, 4, 8, 10]
console.log(even.length); // 4
```

CoGrammar

Maps

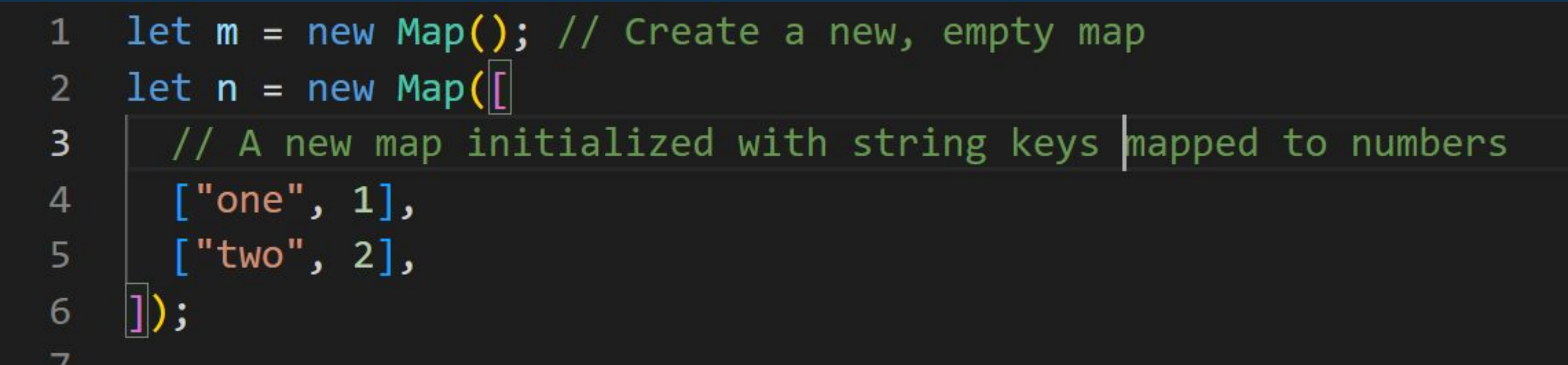
CoGrammar

Maps

A Map object represents a set of values known as keys, where each key has another value associated with (or “mapped to”) it.

❖ In a sense, a **map** is like an array, but instead of using a set of sequential integers as the keys, **maps** allow us to use arbitrary values as **“indexes”.**

❖ You can create a new **map** with the **Map()** constructor.



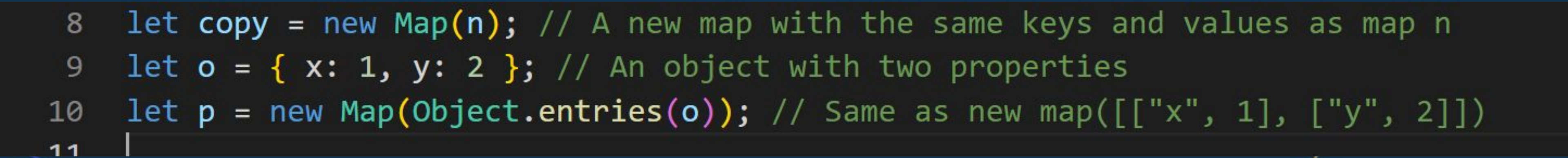
```
let m = new Map(); // Create a new, empty map
let n = new Map([
  // A new map initialized with string keys mapped to numbers
  ["one", 1],
  ["two", 2],
]);
```

Maps

- The optional argument to the **Map()** constructor should be an **iterable** object that yields two element **[key, value] arrays**.
- In practice, this means that if you want to initialize a **map** when you create it, you'll typically write out the desired keys and associated values as an array of arrays.

Maps

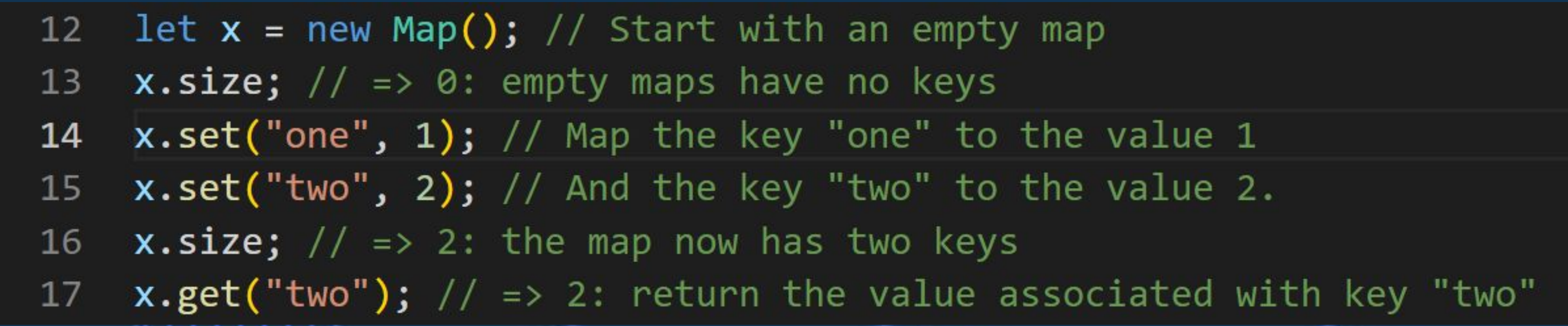
- But you can also use the **Map()** constructor to copy other maps or to copy the property **names** and **values** from an existing object.


```
let copy = new Map(n); // A new map with the same keys and values as map n
let o = { x: 1, y: 2 }; // An object with two properties
let p = new Map(Object.entries(o)); // Same as new map([["x", 1], ["y", 2]])
```

CoGrammar

Maps

- Once you have created a **Map** object, you can query the **value** associated with a given **key** with **get()** and can add a new key/value pair with **set()**.



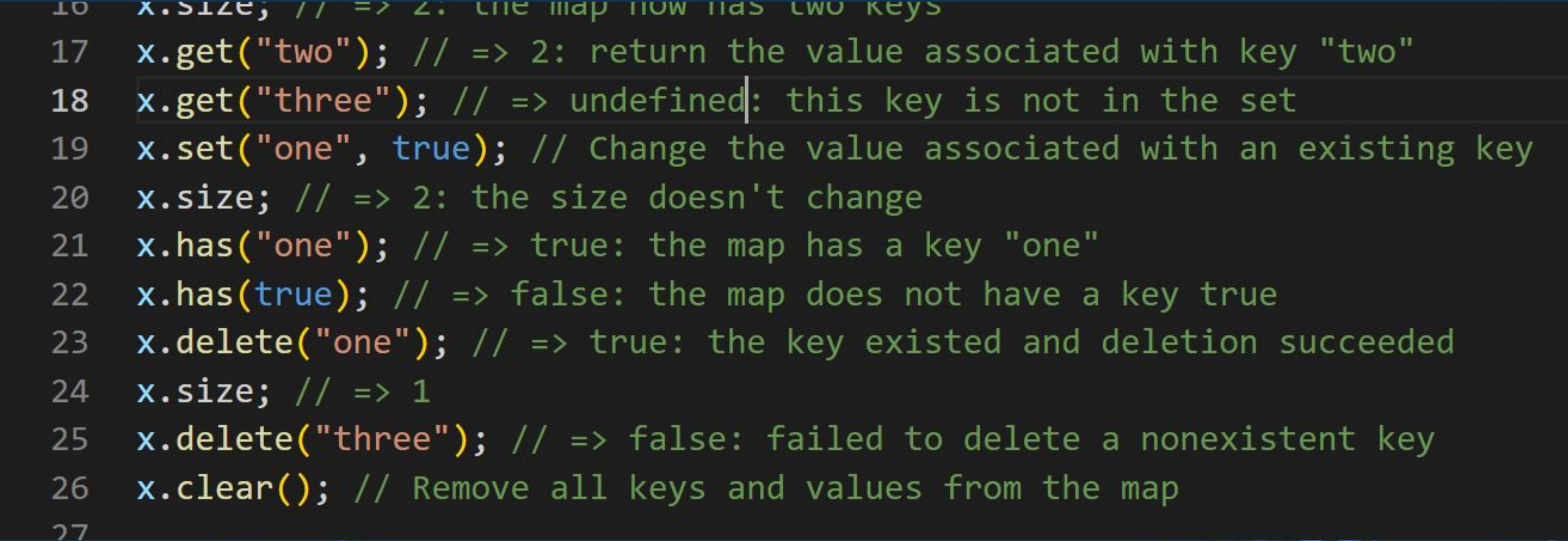
```
let x = new Map(); // Start with an empty map
x.size; // => 0: empty maps have no keys
x.set("one", 1); // Map the key "one" to the value 1
x.set("two", 2); // And the key "two" to the value 2.
x.size; // => 2: the map now has two keys
x.get("two"); // => 2: return the value associated with key "two"
```

Maps

- In addition to **get()** and **set()**,
  - use **has()** to check whether a map includes the specified key
  - use **delete()** to remove a key (and its associated **value**) from the map
  - use **clear()** to remove all **key/value** pairs from the map
  - use the **size** property to find out how many keys a map contains

CoGrammar

Maps

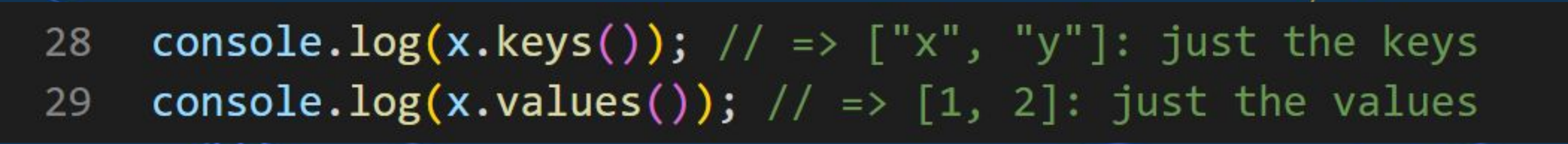


```javascript
x.size; // => 2: the map now has two keys
x.get("two"); // => 2: return the value associated with key "two"
x.get("three"); // => undefined: this key is not in the set
x.set("one", true); // Change the value associated with an existing key
x.size; // => 2: the size doesn't change
x.has("one"); // => true: the map has a key "one"
x.has(true); // => false: the map does not have a key true
x.delete("one"); // => true: the key existed and deletion succeeded
x.size; // => 1
x.delete("three"); // => false: failed to delete a nonexistent key
x.clear(); // Remove all keys and values from the map
```

CoGrammar

Maps

- Map objects are **iterable**, and each **iterated** value is a two-element array where the first element is a **key** and the second element is the **value** associated with that key.
- If you want to **iterate** just the **keys** or just the associated **values** of a **map**, use the **keys()** and **values()** methods: these return iterable objects that iterate keys and values.

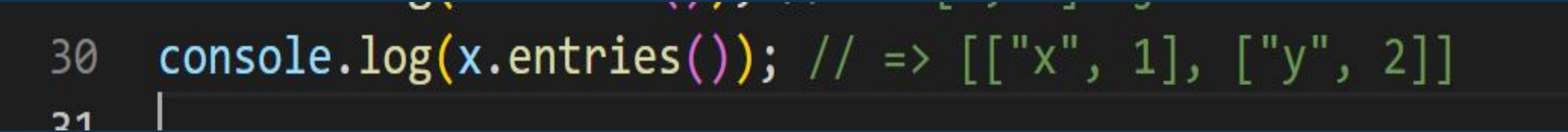


```
console.log(x.keys()); // => ["x", "y"]: just the keys
console.log(x.values()); // => [1, 2]: just the values
```

CoGrammar

Maps

❖ The **entries()** method returns an iterable object that iterates key/value pairs.



```
console.log(x.entries()); // => [["x", 1], ["y", 2]]
```

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar