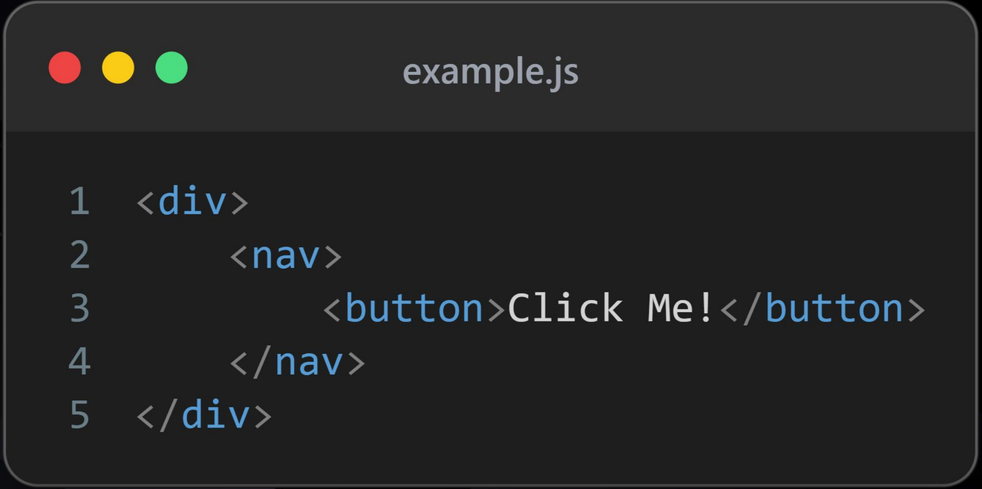
JavaScript

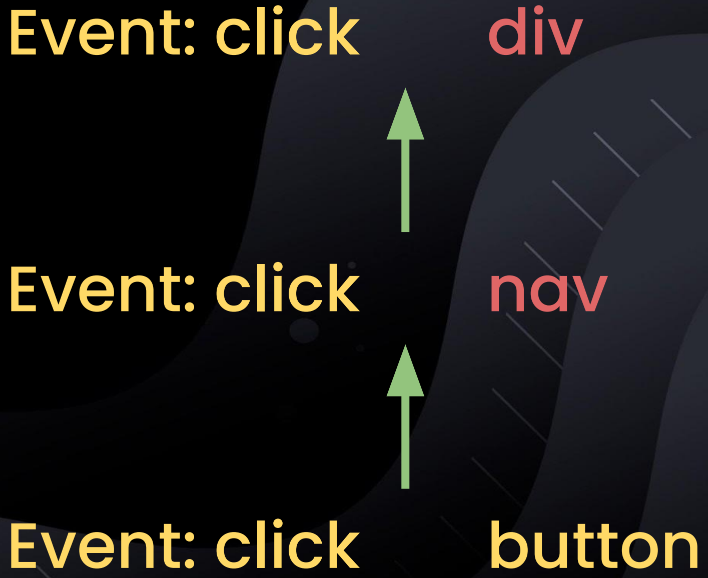
STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA

HOMEWORK

Event Bubbling

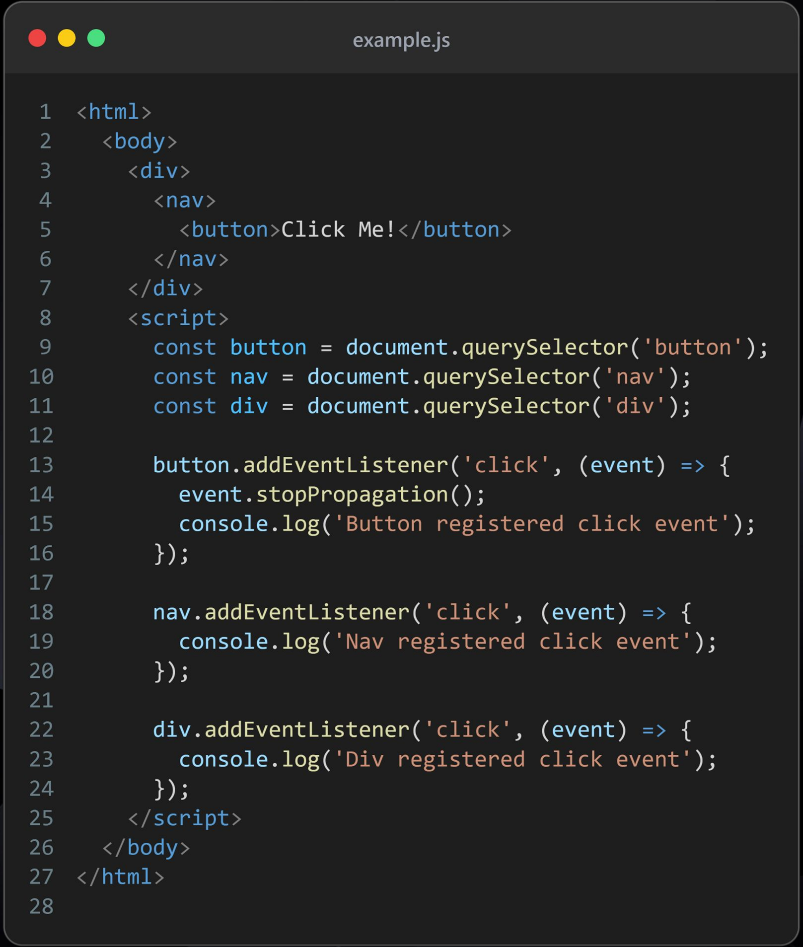

example.js

```
<div>
    <nav>
        <button>Click Me!</button>
    </nav>
</div>
```



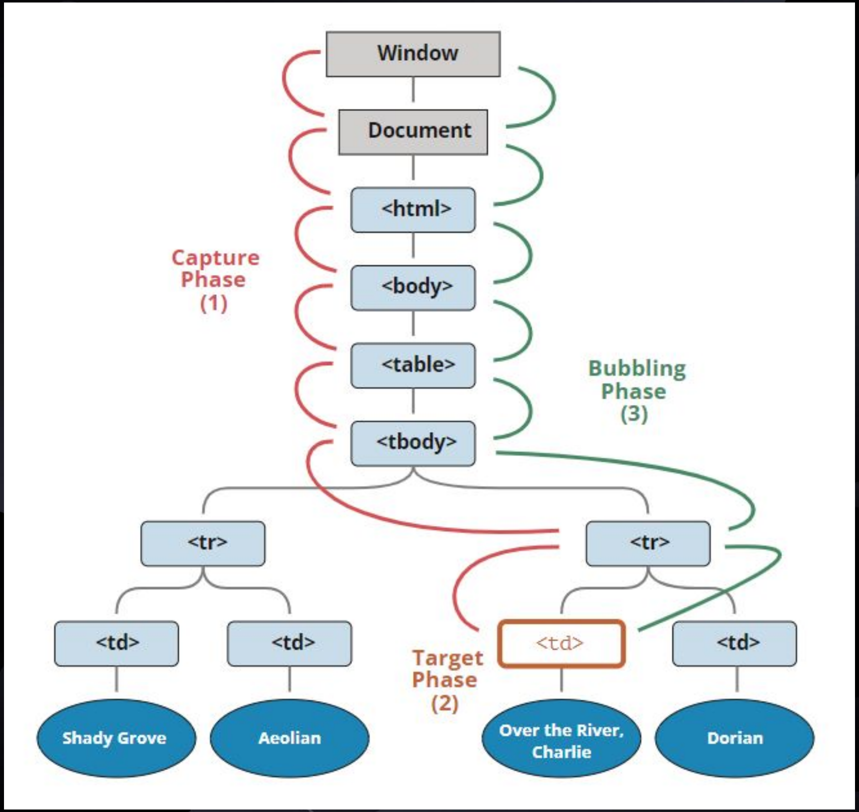
Event Bubbling

- almost all events bubble
- event.target contains the most deeply nested element that caused the event is called
- event.target doesn’t change through the bubbling process
- you can stop bubbling by using event.stopPropagation() method


example.js

```html
<html>
  <body>
    <div>
      <nav>
        <button>Click Me!</button>
      </nav>
    </div>
    <script>
      const button = document.querySelector('button');
      const nav = document.querySelector('nav');
      const div = document.querySelector('div');

      button.addEventListener('click', (event) => {
        event.stopPropagation();
        console.log('Button registered click event');
      });

      nav.addEventListener('click', (event) => {
        console.log('Nav registered click event');
      });

      div.addEventListener('click', (event) => {
        console.log('Div registered click event');
      });
    </script>
  </body>
</html>

```

Event Capturing

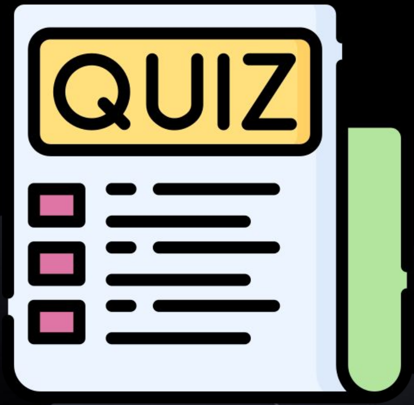


Event Capturing

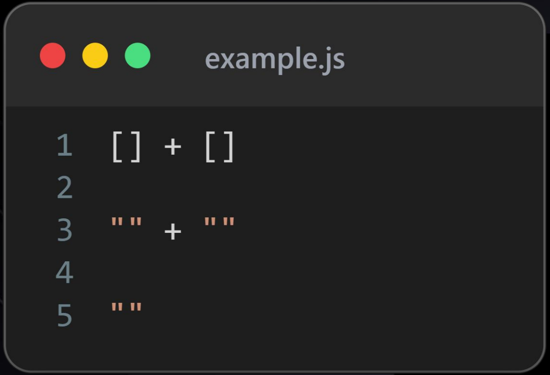
- first of three event phases
- rarely used

QUIZ

type coercion



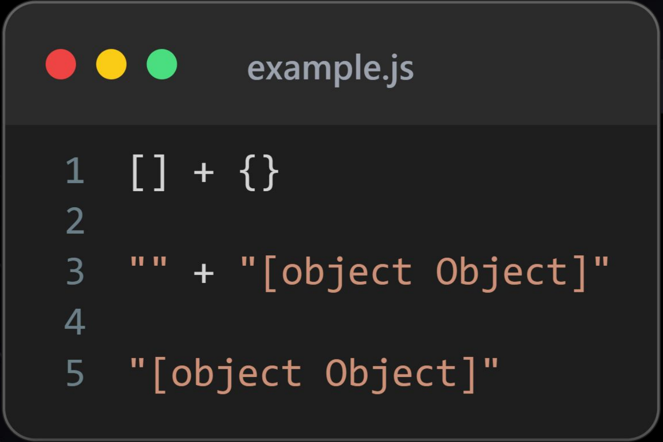
```
[] + []
```



example.js

```
[] + []

"" + ""

""
```

```
[ ] + { }
```

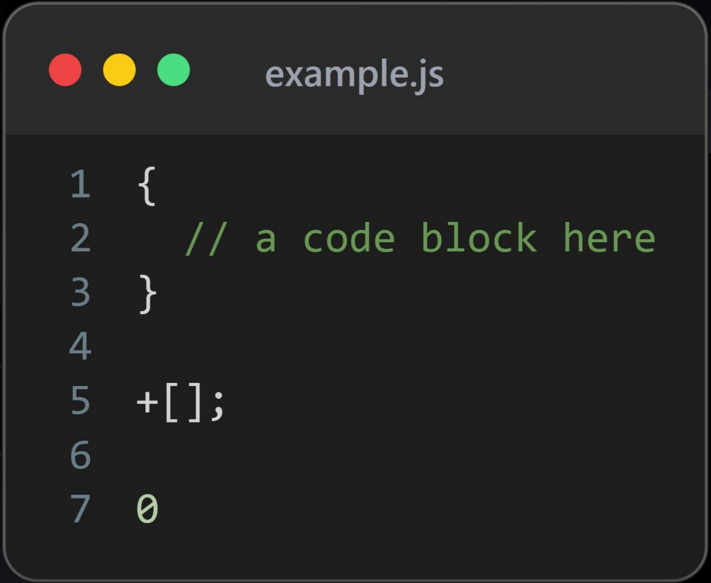


example.js

```
[] + {}

"" + "[object Object]"

"[object Object]"
```

“[object Object]”

```
{} + []
```

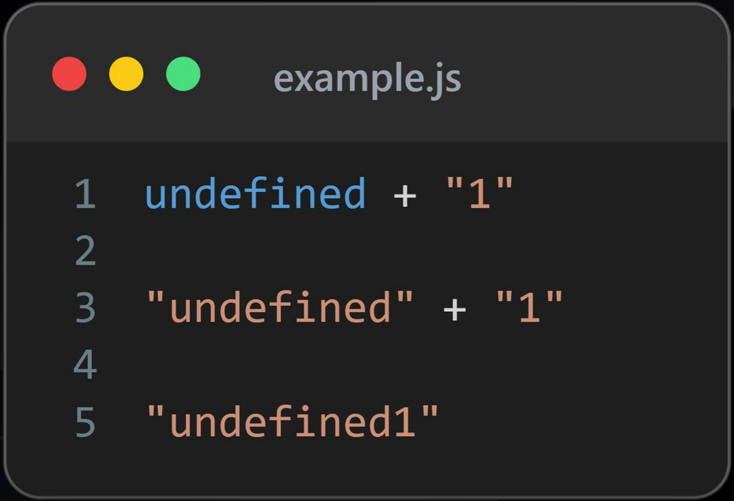


example.js

```js
{
  // a code block here
}

+[];

0
```

0

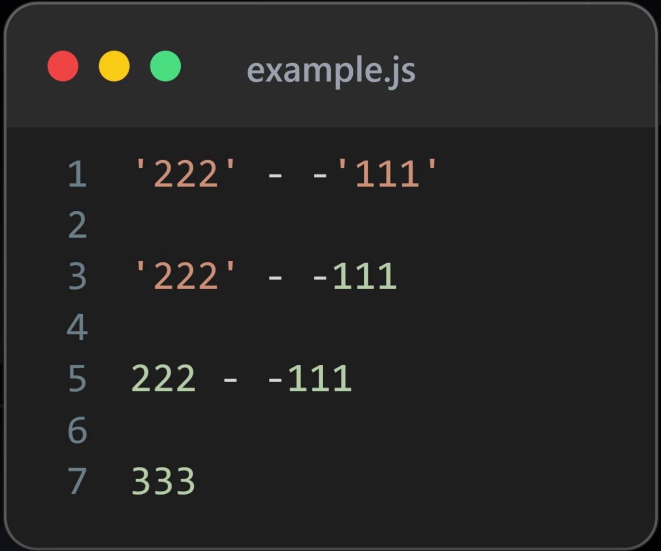
undefined + “1”


example.js

```
undefined + "1"

"undefined" + "1"

"undefined1"
```

“undefined1”

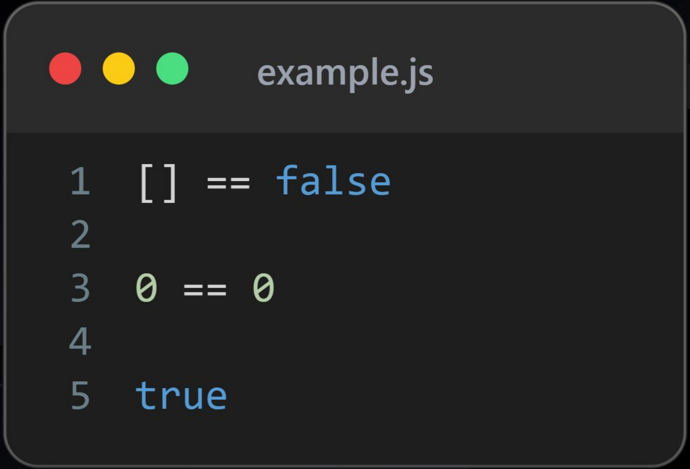
“222” - -“111”


example.js

```js
'222' - -'111'

'222' - -111

222 - -111

333
```

333

```
[ ] == false
```

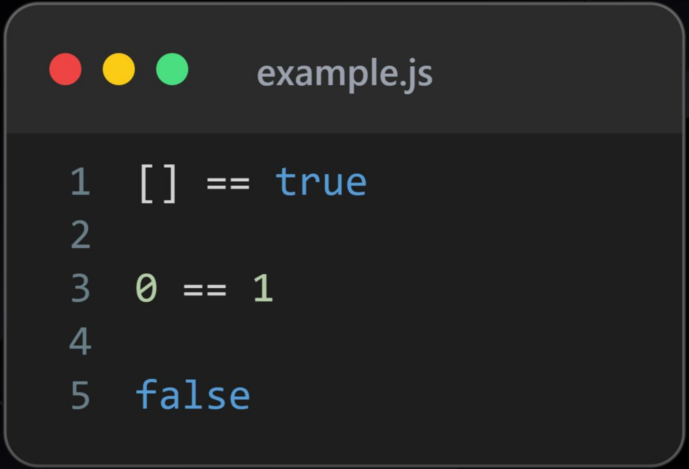


example.js

```
[] == false

0 == 0

true
```

true

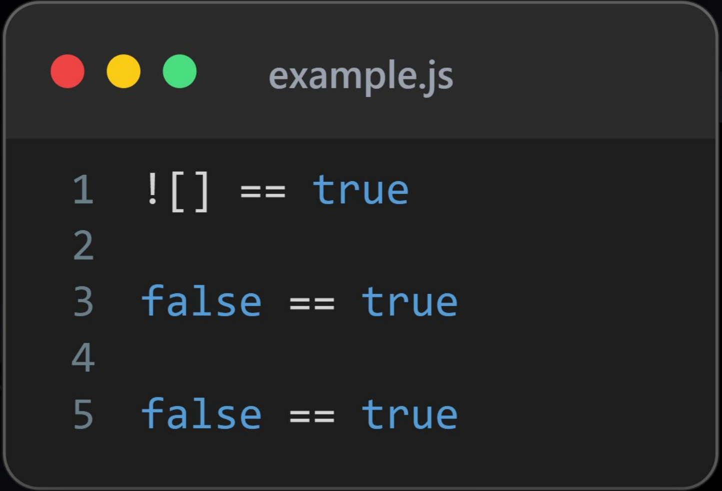
```
[ ] == true
```


example.js

```
[] == true

0 == 1

false
```

false

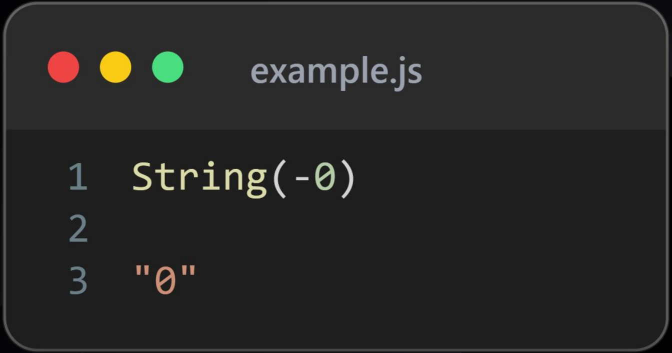
```
![ ] == true
```


example.js

```
![] == true

false == true

false == true
```

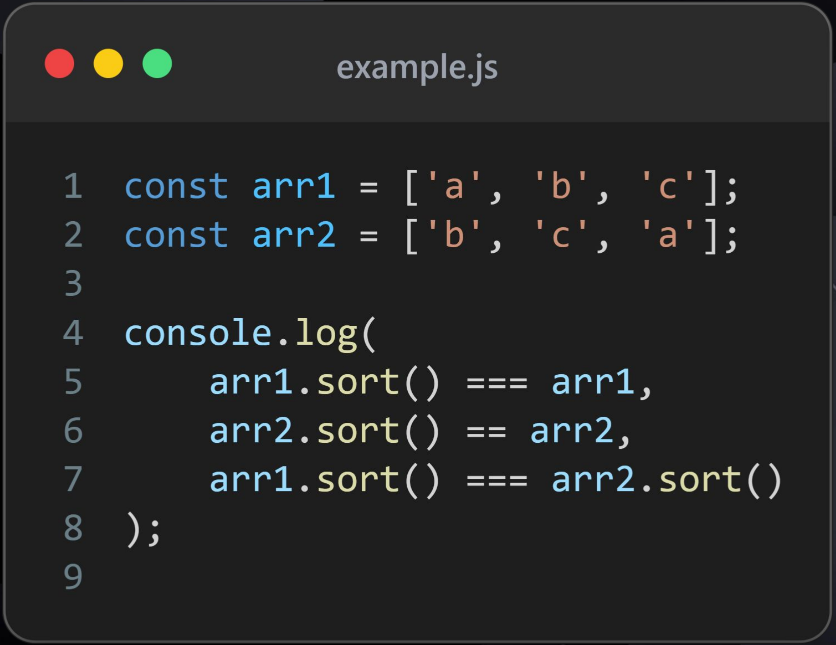
false

```
String(-0)
```


example.js

```
String(-0)

"0"
```

“0”

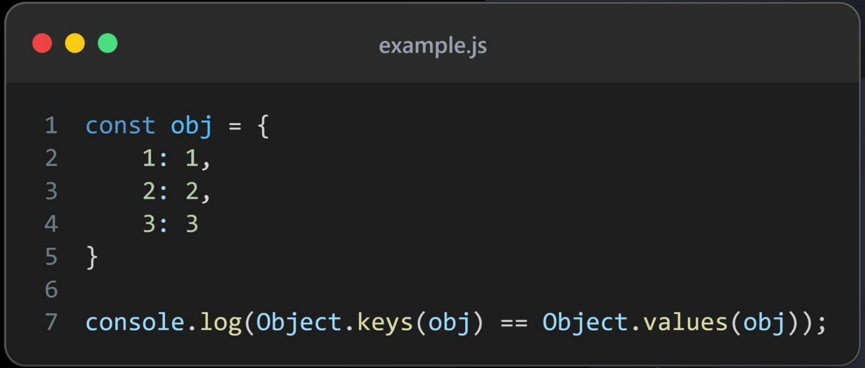


```
const arr1 = ['a', 'b', 'c'];
const arr2 = ['b', 'c', 'a'];

console.log(
    arr1.sort() === arr1,
    arr2.sort() == arr2,
    arr1.sort() === arr2.sort()
);

```

true

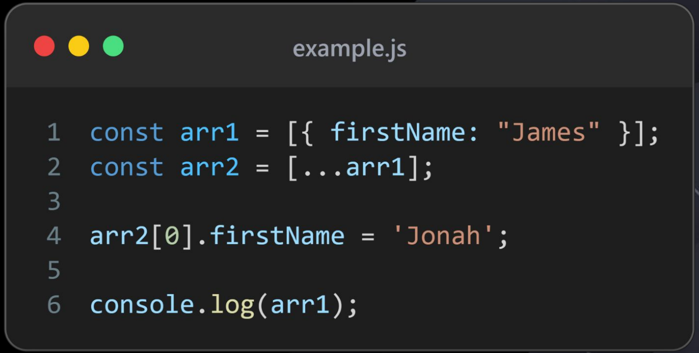
true

false



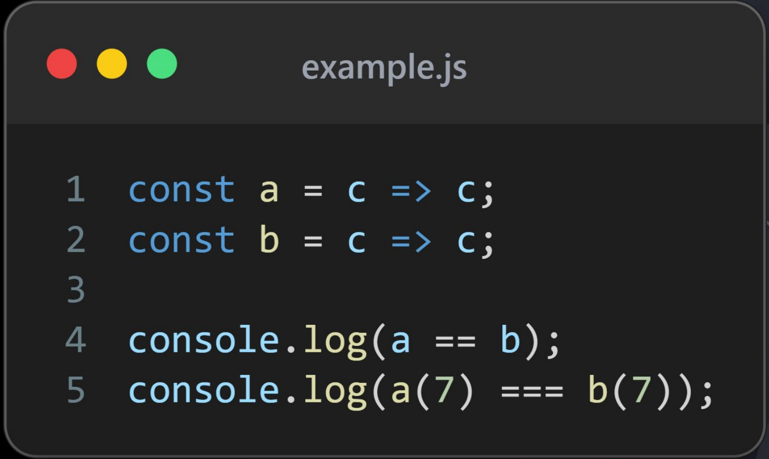
```javascript
const obj = {
    1: 1,
    2: 2,
    3: 3
}

console.log(Object.keys(obj) == Object.values(obj));
```

false



```
const arr1 = [{ firstName: "James" }];
const arr2 = [...arr1];

arr2[0].firstName = 'Jonah';

console.log(arr1);
```

```
[{firstName: 'Jonah'}]
```

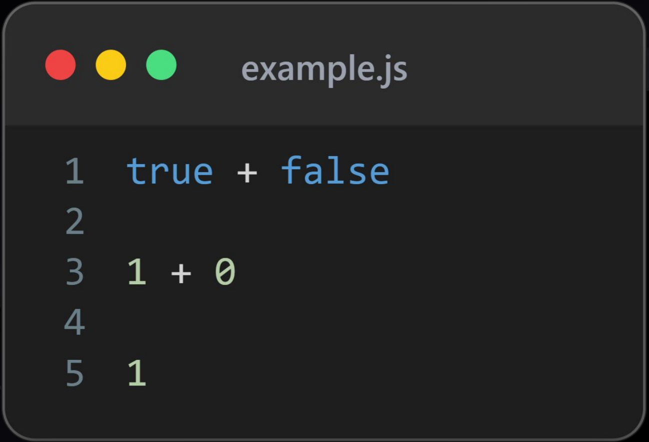

example.js

```
const a = c => c;
const b = c => c;

console.log(a == b);
console.log(a(7) === b(7));
```

false

true

true + false

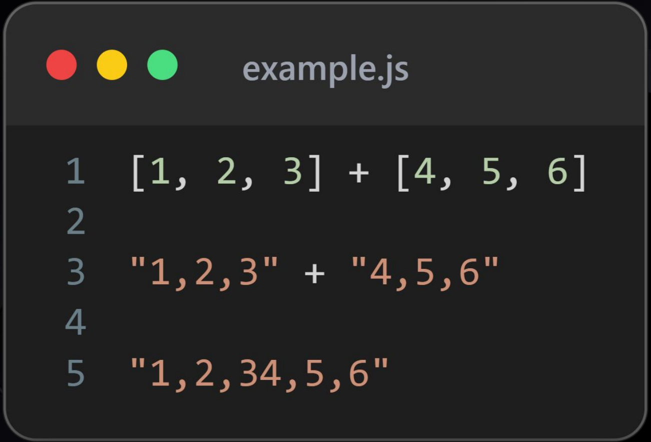


example.js

```
true + false

1 + 0

1
```

1

[1,2,3] + [4,5,6]

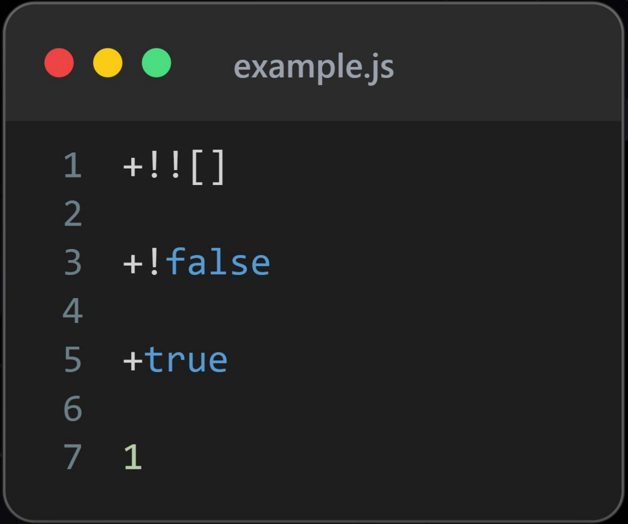


example.js

```
[1, 2, 3] + [4, 5, 6]

"1,2,3" + "4,5,6"

"1,2,34,5,6"
```

“1,2,34,5,6”

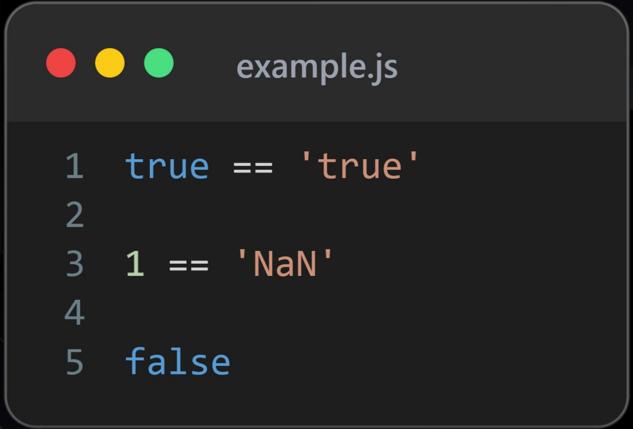
+!![ ]


example.js

```
+!![]

+!false

+true

1
```

1

true == “true”

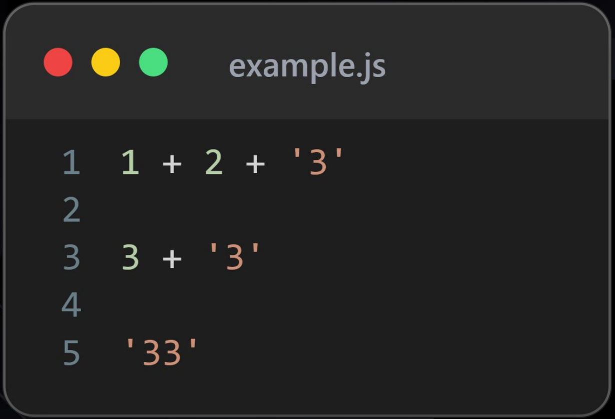


example.js

```
true == 'true'

1 == 'NaN'

false
```

false

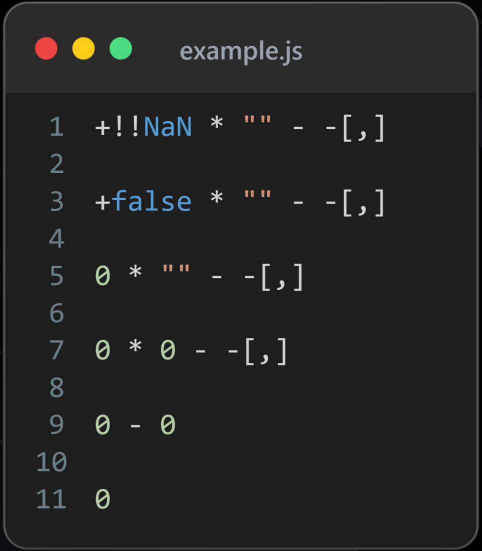
1 + 2 + “3”



```
1 + 2 + '3'

3 + '3'

'33'
```

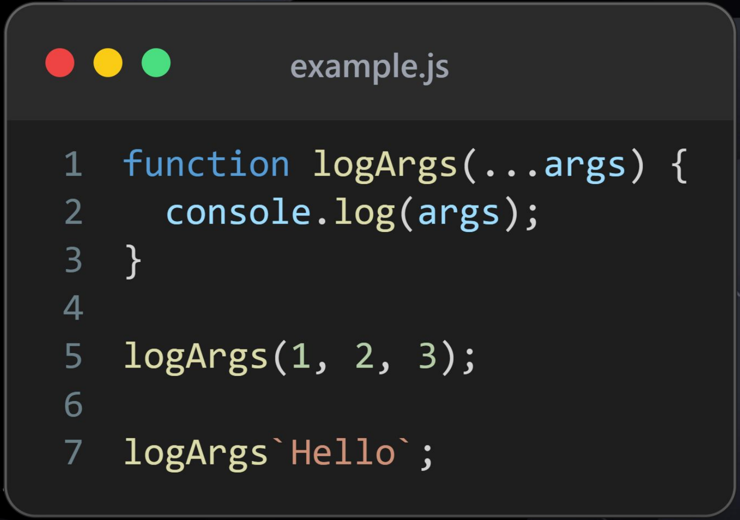
"33"

```
+!!NaN * “”- - [,]
```


example.js

```
+!!NaN * "" - -[,]

+false * "" - -[,]

0 * "" - -[,]

0 * 0 - -[,]

0 - 0

0
```

0

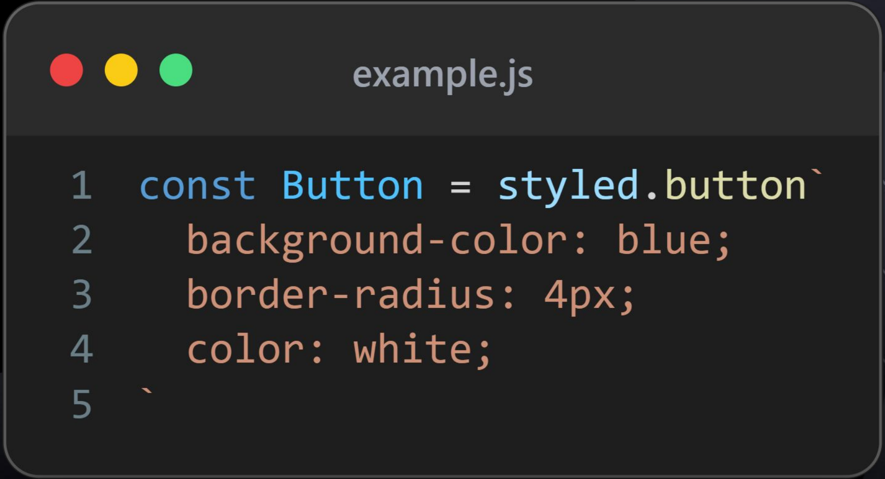

example.js

```
function logArgs(...args) {
  console.log(args);
}

logArgs(1, 2, 3);

logArgs`Hello`;
```

```
[1, 2, 3]
[[“Hello”]]
```

Tagged template literal

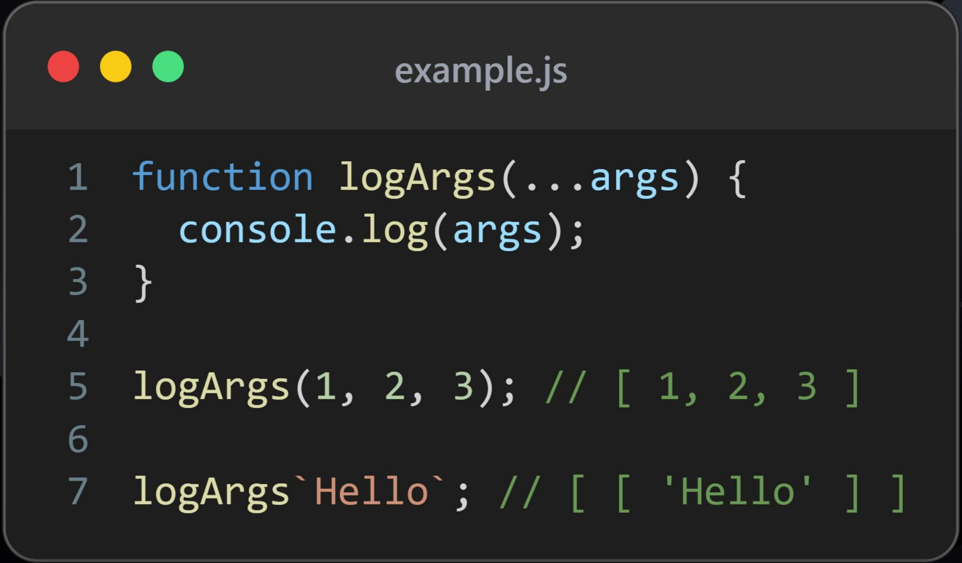
styled components



example.js

```
const Button = styled.button`
  background-color: blue;
  border-radius: 4px;
  color: white;
`
```

Tagged template literal

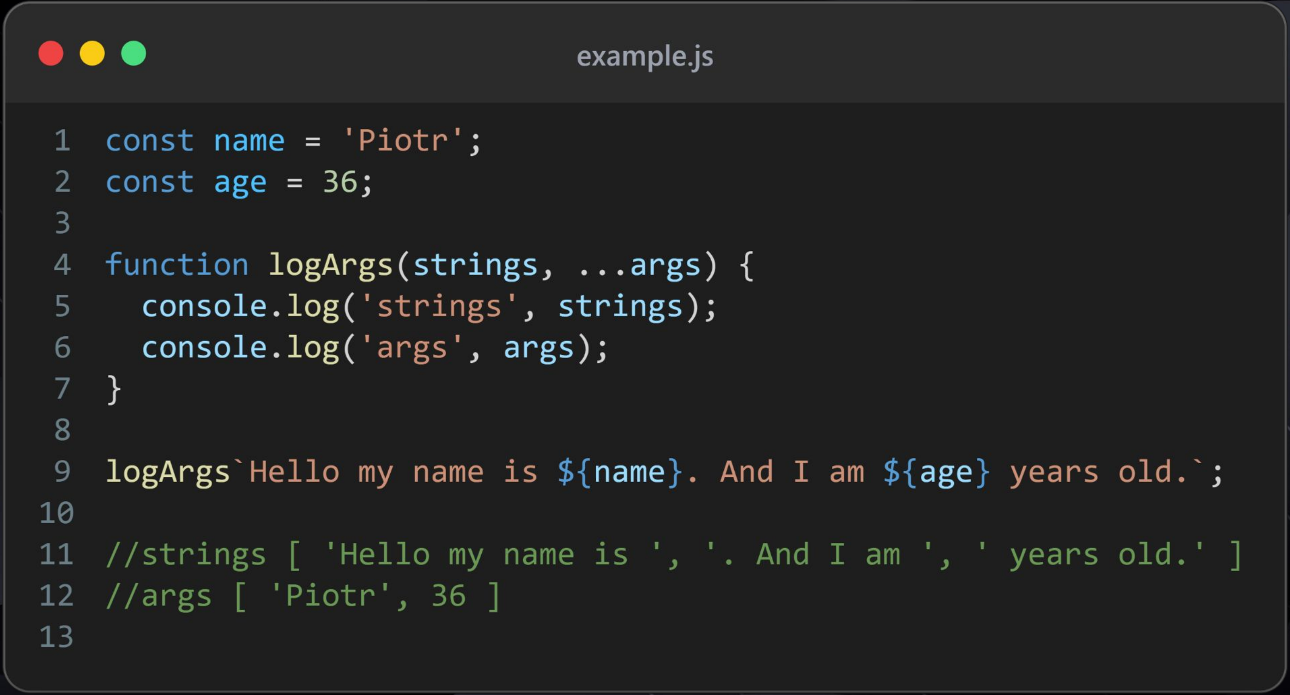


```javascript
function logArgs(...args) {
  console.log(args);
}

logArgs(1, 2, 3); // [ 1, 2, 3 ]

logArgs`Hello`; // [ [ 'Hello' ] ]
```

Tagged template literal

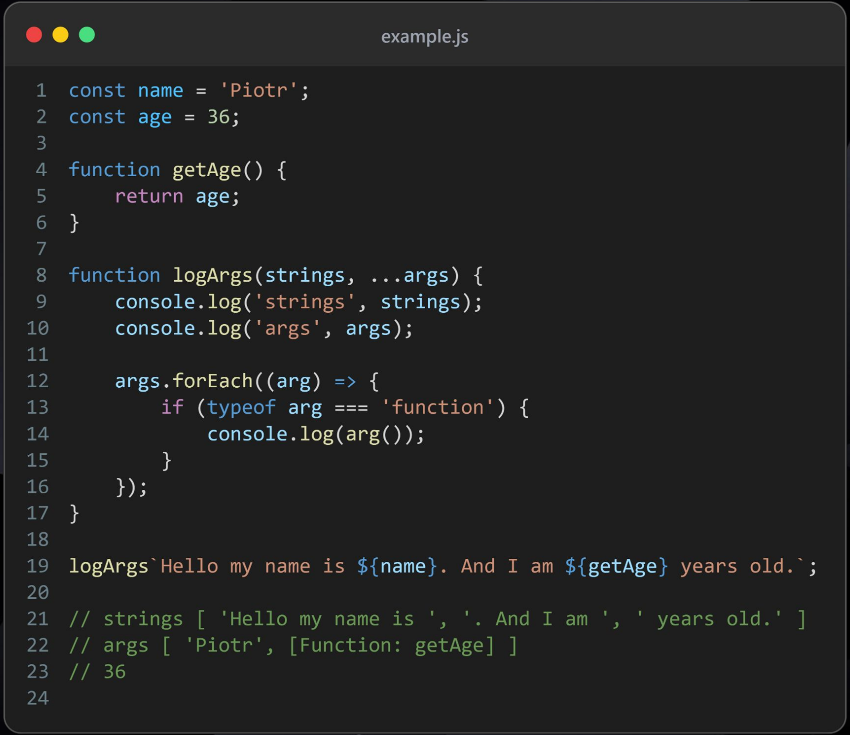
- Allows you to add template literal to function
- Tags allows function to parse template literals
- first argument of a tag function contains an array of string values
- remaining arguments are expressions used in template

Tagged template literal



```javascript
const name = 'Piotr';
const age = 36;

function logArgs(strings, ...args) {
  console.log('strings', strings);
  console.log('args', args);
}

logArgs`Hello my name is ${name}. And I am ${age} years old.`;

//strings [ 'Hello my name is ', '. And I am ', ' years old.' ]
//args [ 'Piotr', 36 ]

```

Tagged template literal

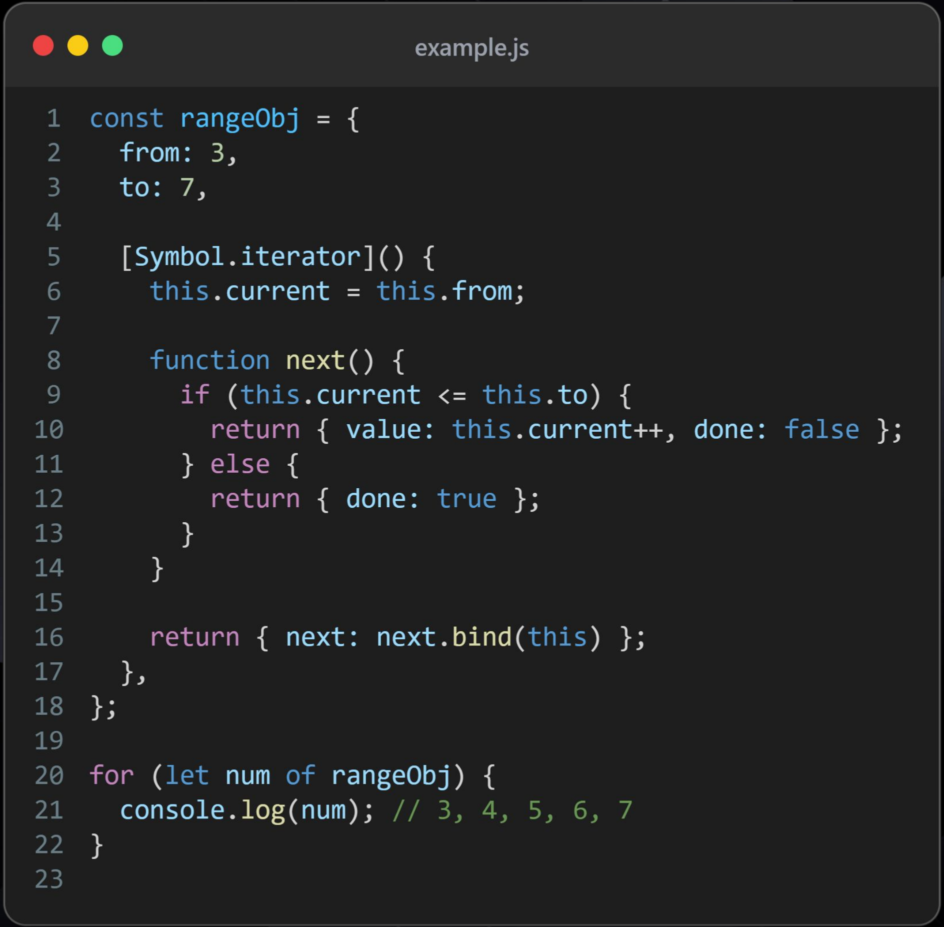


example.js

```javascript
const name = 'Piotr';
const age = 36;

function getAge() {
    return age;
}

function logArgs(strings, ...args) {
    console.log('strings', strings);
    console.log('args', args);

    args.forEach((arg) => {
        if (typeof arg === 'function') {
            console.log(arg());
        }
    });
}

logArgs`Hello my name is ${name}. And I am ${getAge} years old.`;

// strings [ 'Hello my name is ', '. And I am ', ' years old.' ]
// args [ 'Piotr', [Function: getAge] ]
// 36

```

Iterator

- Create object with to properties: from and to
- Create iterator that will iterate based on the given range



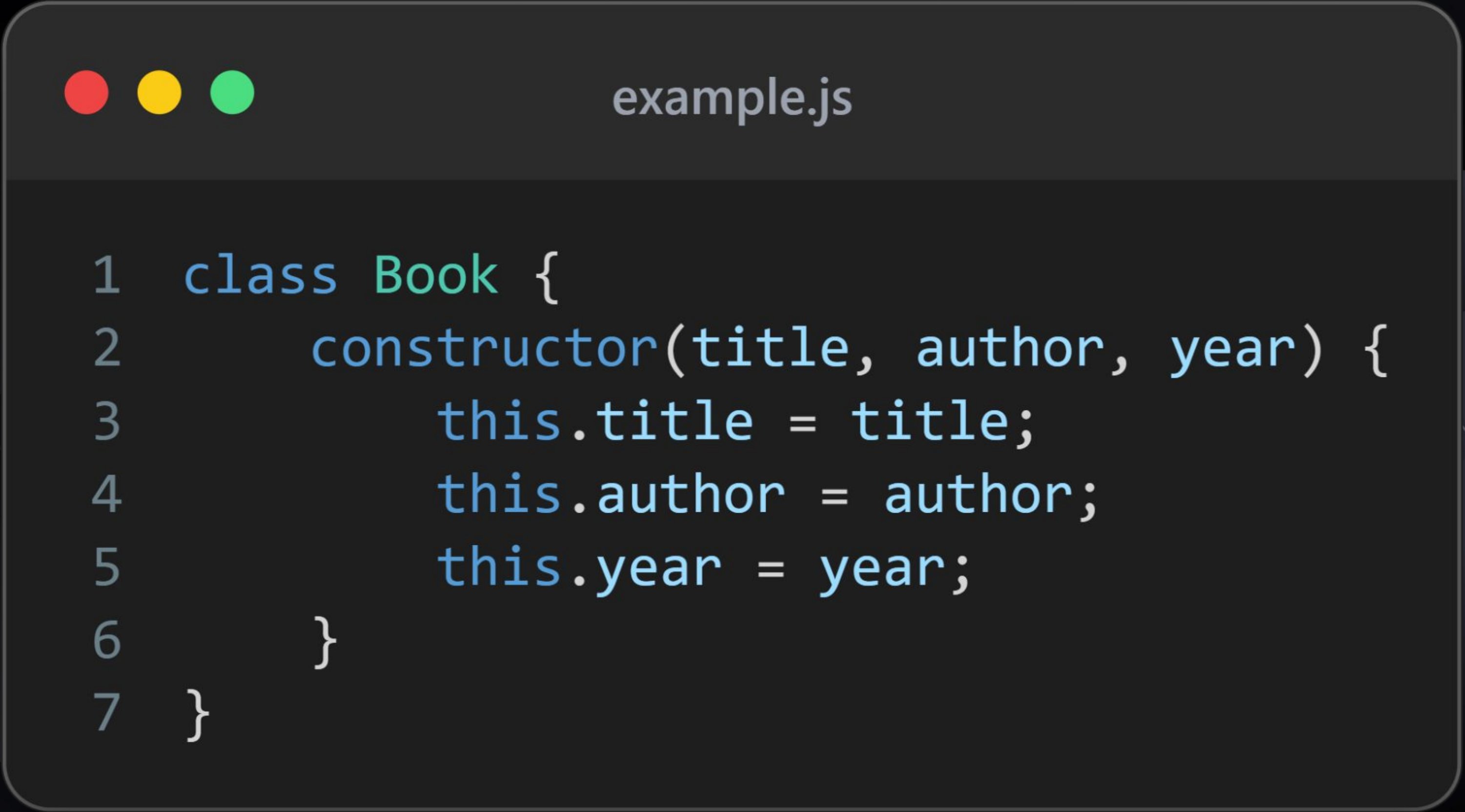
```javascript
const rangeObj = {
  from: 3,
  to: 7,

  [Symbol.iterator]() {
    this.current = this.from;

    function next() {
      if (this.current <= this.to) {
        return { value: this.current++, done: false };
      } else {
        return { done: true };
      }
    }

    return { next: next.bind(this) };
  },
};

for (let num of rangeObj) {
  console.log(num); // 3, 4, 5, 6, 7
}

```

Iterator

- Create a library that implements an iterator that returns each book stored in the library’s catalog
- Using the iterator create a consuming function that allow one to rate a book

Iterator

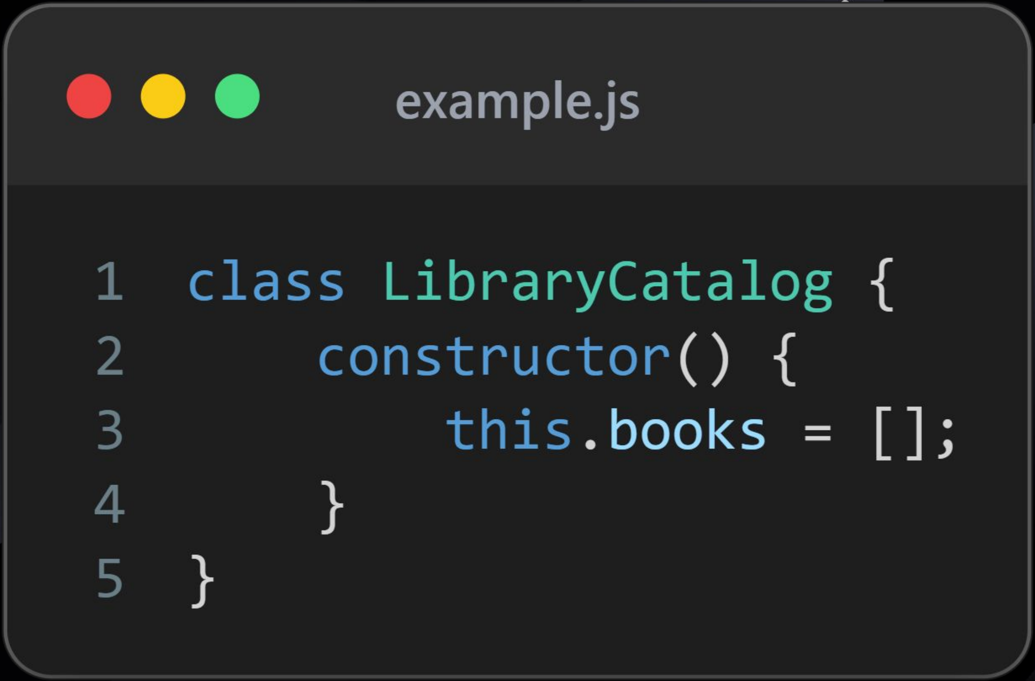
- Define a class called Book with the following properties:
    - title (string) - Book title
    - author (string) - Book author
    - year (number) - Publication year


example.js

```javascript
class Book {
    constructor(title, author, year) {
        this.title = title;
        this.author = author;
        this.year = year;
    }
}
```

Iterator

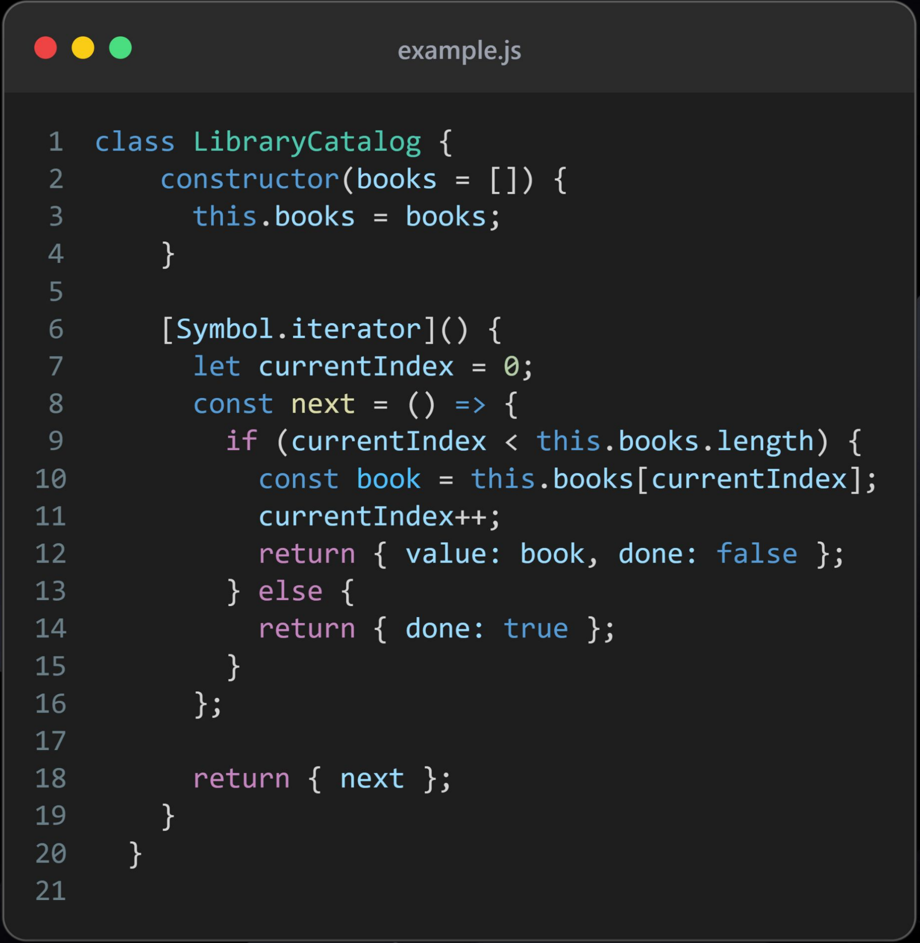
- Define a class called LibraryCatalog that has the following properties:
  - books (array): An array of Book objects



```
class LibraryCatalog {
    constructor() {
        this.books = [];
    }
}
```

Iterator

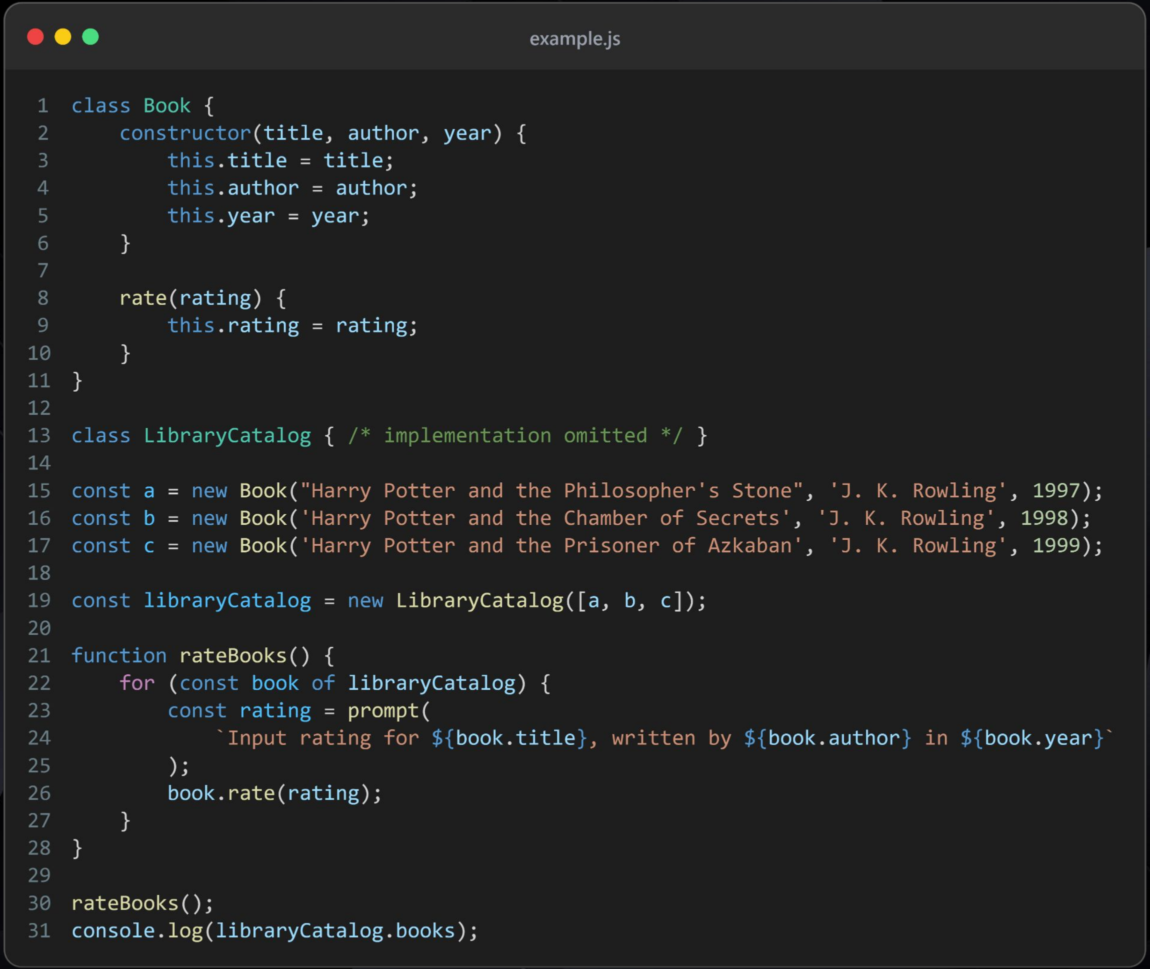
- Implement a custom iterator for the LibraryCatalog class. The iterator should allow iterating over the books in the catalog.
- The iterator should have the following methods:
  - next(): Returns the next book in the catalog. If there are no more books, it should return {done: true}



```javascript
class LibraryCatalog {
    constructor(books = []) {
      this.books = books;
    }

    [Symbol.iterator]() {
      let currentIndex = 0;
      const next = () => {
        if (currentIndex < this.books.length) {
          const book = this.books[currentIndex];
          currentIndex++;
          return { value: book, done: false };
        } else {
          return { done: true };
        }
      };

      return { next };
    }
  }

```

Iterator

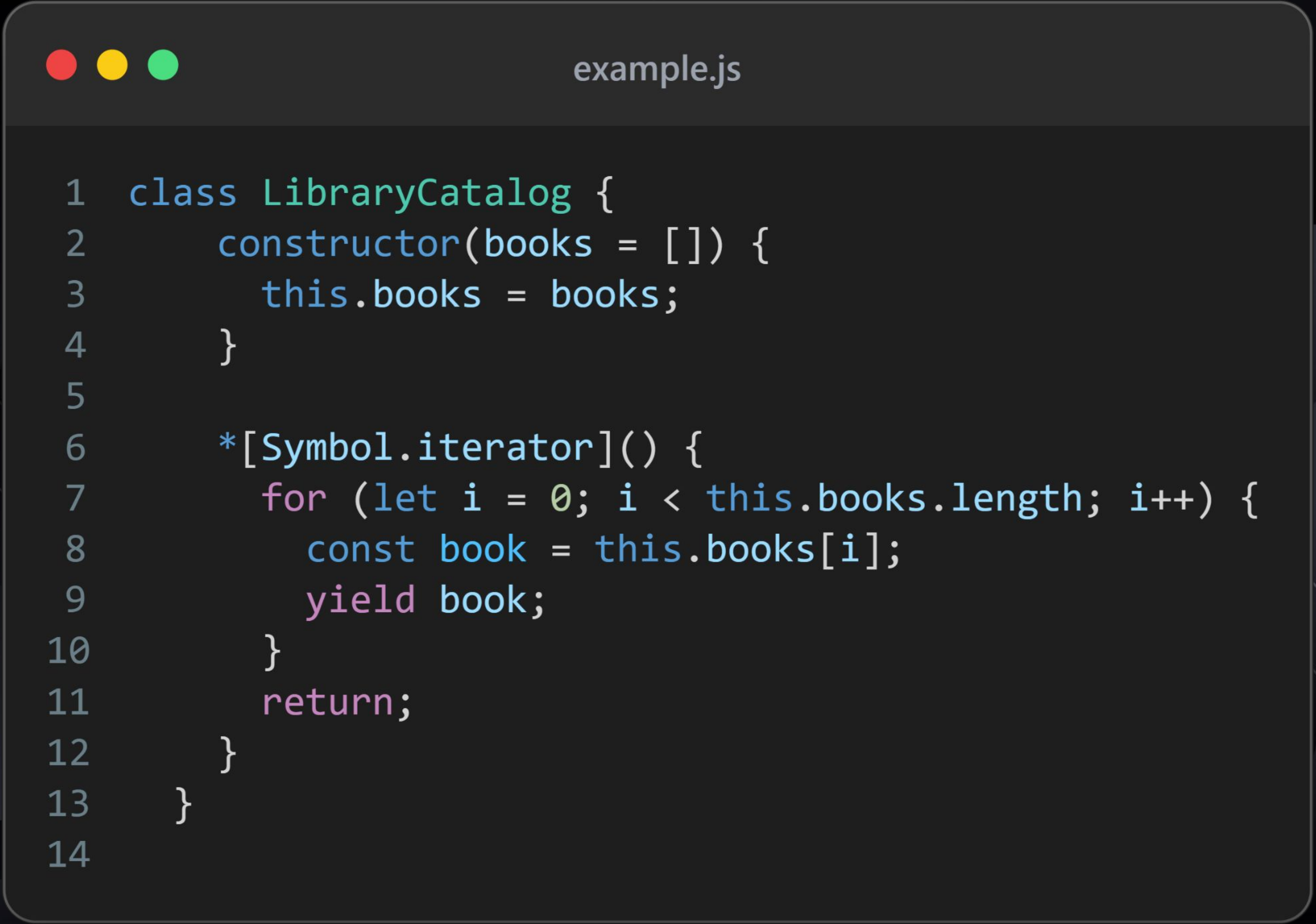
- Create a function that prompts the user to rate each book in the library catalog


example.js

```javascript
class Book {
    constructor(title, author, year) {
        this.title = title;
        this.author = author;
        this.year = year;
    }

    rate(rating) {
        this.rating = rating;
    }
}

class LibraryCatalog { /* implementation omitted */ }

const a = new Book("Harry Potter and the Philosopher's Stone", 'J. K. Rowling', 1997);
const b = new Book('Harry Potter and the Chamber of Secrets', 'J. K. Rowling', 1998);
const c = new Book('Harry Potter and the Prisoner of Azkaban', 'J. K. Rowling', 1999);

const libraryCatalog = new LibraryCatalog([a, b, c]);

function rateBooks() {
    for (const book of libraryCatalog) {
        const rating = prompt(
            `Input rating for ${book.title}, written by ${book.author} in ${book.year}`
        );
        book.rate(rating);
    }
}

rateBooks();
console.log(libraryCatalog.books);
```

Iterator

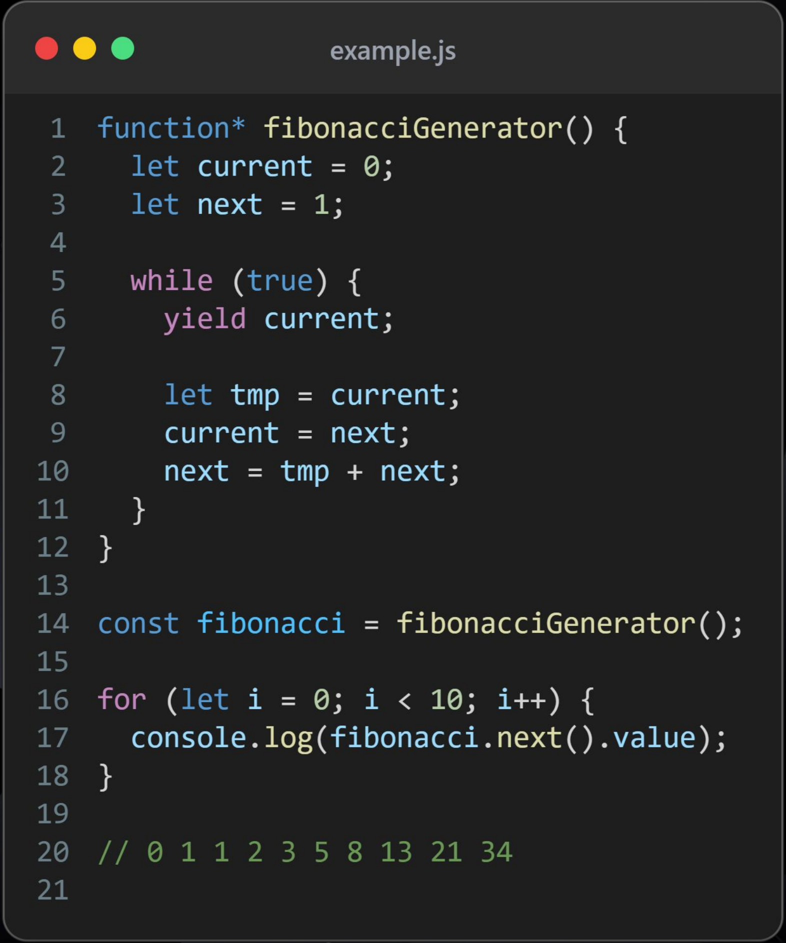
- Refactor iterator to use a generator function


example.js

```javascript
class LibraryCatalog {
    constructor(books = []) {
      this.books = books;
    }

    *[Symbol.iterator]() {
      for (let i = 0; i < this.books.length; i++) {
        const book = this.books[i];
        yield book;
      }
      return;
    }
  }

```

Generator

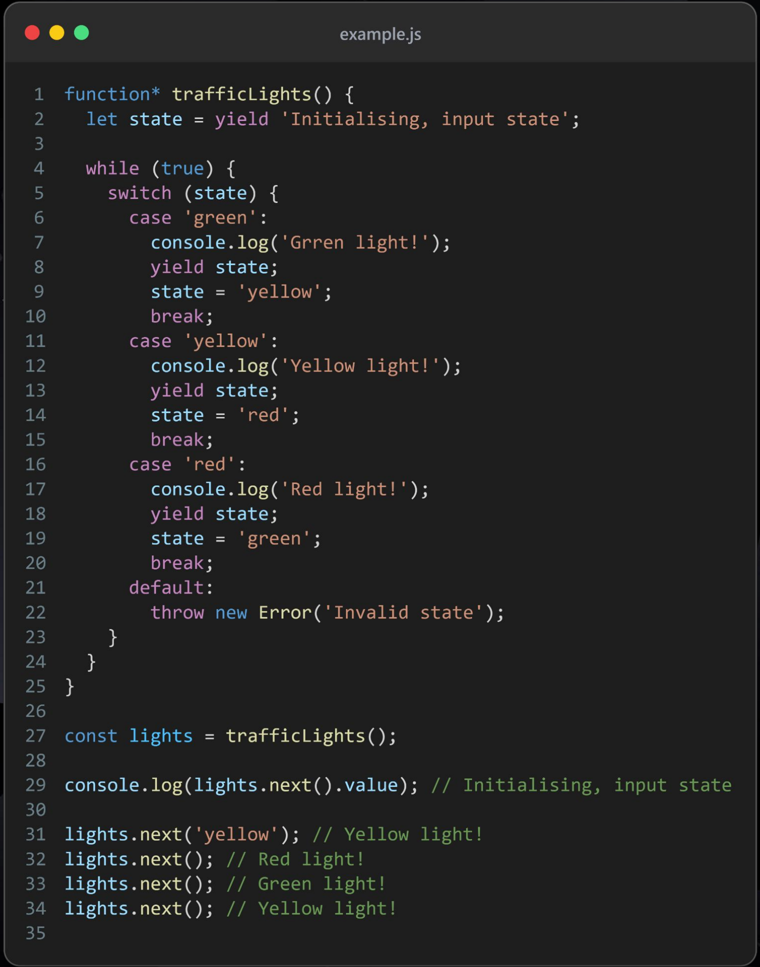
- Implement fibonacci generator


example.js

```
function* fibonacciGenerator() {
  let current = 0;
  let next = 1;

  while (true) {
    yield current;

    let tmp = current;
    current = next;
    next = tmp + next;
  }
}

const fibonacci = fibonacciGenerator();

for (let i = 0; i < 10; i++) {
  console.log(fibonacci.next().value);
}

// 0 1 1 2 3 5 8 13 21 34

```

Generator

- Implement traffic lights using a generator function
- Use state machine pattern
- The state machine represents a traffic light system with three states: “green”, “yellow” and “red”
- The state machine transitions from one state to another based on the input passed to the generator function using the yield statement


example.js

```javascript
function* trafficLights() {
  let state = yield 'Initialising, input state';

  while (true) {
    switch (state) {
      case 'green':
        console.log('Grren light!');
        yield state;
        state = 'yellow';
        break;
      case 'yellow':
        console.log('Yellow light!');
        yield state;
        state = 'red';
        break;
      case 'red':
        console.log('Red light!');
        yield state;
        state = 'green';
        break;
      default:
        throw new Error('Invalid state');
    }
  }
}

const lights = trafficLights();

console.log(lights.next().value); // Initialising, input state

lights.next('yellow'); // Yellow light!
lights.next(); // Red light!
lights.next(); // Green light!
lights.next(); // Yellow light!

```