




JavaScript

STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA

The image features a dark, abstract background with wavy, organic shapes in shades of dark blue and black. Scattered throughout are small white dots, resembling stars or distant galaxies. A prominent, glowing yellow "#1" is centered in the image, with a soft yellow glow radiating from it. The overall aesthetic is modern and minimalist, suggesting a theme of achievement or ranking.

#1

Semester Overview

1. JS in depth – this, global, strict
2. JS in depth – prototypal inheritance
3. Clean code
4. Design Patterns 1
5. Design Patterns 2
6. Iterators, generators, implicit coercion
7. Full-stack project setup

GLOBAL



Global scope/context

- The context that the whole code is run in
- It has global memory
- Variables in the global memory are available throughout your program
- The variables in global memory are **scoped globally**

global.js

```
1 // Global
2
3 const globalVariable = 'My variable';
4
5 console.log('Global scope', globalVariable); // "My variable"
6
7 if(true) {
8     console.log('Block scope', globalVariable); // "My variable"
9 }
10
11 function logGlobalVar() {
12     console.log('Function scope', globalVariable);
13 }
14
15 logGlobalVar(); // "My variable"
```

Global Object

- Object that exists for every JS runtime in global memory
- It stores all utility functions and objects
- Depending on runtime named differently (global, window, frames, WorkGlobalScope, self)
- Since ES2020 unified under globalThis



global.js

```
1  // Global
2
3  global.console.log('hello');
4
5  const today = new global.Date();
6
7  console.log(today);
8
9  // Node ENV
10 console.log(globalThis === global);
11
12 // Browser ENV
13 console.log(globalThis === window);
14
```


this

- defaults to the global object (in non-strict mode)
- not consistent behavior – **this** in the global context is an empty object in Node
- **this** can cause some weird bugs!
Especially in functions

this.js

```
1 // This - defaults to global in BROWSER (in non-strict mode)
2
3 console.log(this);
4
5 console.log(this.Date.now());
6
```

this.js

```
1  // This - defaults to global in functions context (all environments)
2
3  const transaction = {
4    amount: 1000,
5    updateAmount(value) {
6      this.amount = value;
7    },
8  };
9
10 function runFunctionWithTime(callback, ...args) {
11   const timeStamp = Date.now();
12   console.log('Running function at: ', timeStamp);
13   callback(...args);
14 };
15
16 runFunctionWithTime(transaction.updateAmount, 2000);
17
18 console.log(transaction.amount); // 1000
19 console.log(globalThis.amount); // 2000 !
20
```

strict mode

- mode that can be run globally or in a function
- all code inside native modules and classes is run in strict mode!
- in production env rarely is needed since we work in modules and our code is usually transpiled and there's a linter

strict.js

```
1 // Strict mode - this defaults to undefined in functions
2
3 'use strict';
4
5 const transaction = {
6   amount: 1000,
7   updateAmount(value) {
8     this.amount = value;
9   },
10 };
11
12 function runFunctionWithTime(callback, ...args) {
13   const timeStamp = Date.now();
14   console.log('Running function at: ', timeStamp);
15   callback(...args);
16 }
17
18 runFunctionWithTime(transaction.updateAmount, 2000);
19 // Cannot set properties of undefined (setting 'amount')
20
```

strict.js

```
1  // No keyword declaration
2
3  function logArray(inputArray) {
4      for (i = 0; i < inputArray.length; i++) {
5          console.log(inputArray[i]);
6      }
7  };
8
9  const hobbits = ['Frodo', 'Merry', 'Sam', 'Pippin'];
10 logArray(hobbits);
11
12 console.log(i); // 4 !
13 console.log(globalThis.i); // 4 !
14
```

```
1 // Strict mode - no keyword declaration
2 'use strict';
3
4 function logArray(inputArray) {
5     for (i = 0; i < inputArray.length; i++) { //ReferenceError: i is not defined
6         console.log(inputArray[i]);
7     }
8 };
9
10 const hobbits = ['Frodo', 'Merry', 'Sam', 'Pippin'];
11 logArray(hobbits);
12
13 console.log(globalThis.i); // undefined
14 console.log(i); // error
15
```

strict mode

[Strict Mode MDN](#)

Let's get back to **this**

- 'The value of this depends on in which context it appears: function, class or global.' – **what does it even mean?**
- **this** depends only on **HOW** our function is run, not **WHERE** it was declared

EXCEPTION

Arrow functions and **this**

- Arrow functions don't create **this** in their context
- **this** in arrow function depends on where the function was declared (lexical scope)



this.js

```
1  // Arrow functions and this keyword
2
3  const test = {
4    name: 'test obj',
5    logName: () => {
6      console.log(this.name);
7    },
8  };
9
10 test.logName(); // undefined
```

this.js

```
1  // Arrow functions and this keyword
2
3  const test = {
4    name: 'test obj',
5    getThisLogger() {
6      return () => {
7        console.log(this);
8      };
9    },
10 };
11
12 const logThis = test.getThisLogger();
13
14 logThis(); // {name: 'test obj', getThisLogger: f}
15
```

this.js

```
1 // Implicit binding
2
3 const student = {
4   name: 'Bob',
5   surname: 'The Builder',
6   year: 1,
7   logStudent() {
8     console.log(`${this.name} ${this.surname} is in the ${this.year} year`);
9   },
10 };
11
12 student.logStudent(); // Bob The Builder is in the 1 year
```

this.js

```
1  // New keyword
2
3  function studentCreator(name, surname) {
4    this.name = name;
5    this.surname = surname;
6  }
7
8  const student = new studentCreator('Bob', 'The Builder');
9  console.log(student); // {name: 'Bob', surname: 'The Builder'}
10
```

this.js

```
1  // Explicit binding
2
3  const student = {
4    name: 'Bob',
5    surname: 'The Builder',
6  };
7
8  function updateStudent(name, surname) {
9    this.name = name;
10   this.surname = surname;
11  }
12
13  console.log(student); // {name: 'Bob', surname: 'The Builder'}
14
15  updateStudent.apply(student, ['Dora', 'The Explorer']);
16  console.log(student); // {name: 'Dora', surname: 'The Explorer'}
17
18  updateStudent.call(student, 'Pat', 'The Postman');
19  console.log(student); // {name: 'Pat', surname: 'The Postman'}
20
```



this.js

```
1 // Hard binding
2
3 const studentOne = {
4   name: 'Bob',
5   surname: 'The Builder',
6 };
7
8 const studentTwo = {
9   name: 'Dora',
10  surname: 'The Explorer',
11 };
12
13 function logStudent() {
14   console.log(this.name, this.surname);
15 }
16
17 const boundLogStudent = logStudent.bind(studentOne);
18
19 boundLogStudent(); // Bob The Builder
20
21 studentTwo.log = boundLogStudent;
22 studentTwo.log(); // Bob The Builder
23 boundLogStudent.call(studentTwo) // Bob The Builder
24
```


Things influencing **this**

- implicit binding (dot - `'.'`)
- `new`
- `call/apply` (explicit binding)
- `bind` (hard binding)

ES modules (native)

- import
- export
- imply strict mode

importing

- default importing
- named import
- namespace import
- side effect import

import.js

```
1 // ESM - import
2
3 import axios from 'axios'; // default import
4
5 import {map} from 'underscore'; // named import (import selected functions/variables)
6
7 import * as React from 'react'; // namespace import (import everything as)
8
9 import './initialization.js'; // side effect import
10
11 // Renaming import
12 import axios as apiClient from 'axios';
13
```

exporting

- export declaration
- export list
- export default
- aggregating exports

export.js

```
1  // ESM - export
2
3  // Export declaration
4  export const student = { name: 'Bob' };
5
6  // Export list
7  const studentTwo = { name: 'Kate' };
8  export { student, studentTwo };
9
10 // Default export
11 export default studentTwo;
12
13 // Aggregate export
14 export { studentThree, studentFour } from './exports.js'
15
```

import-export.js

```
1 // ESM - export and imports - how it works
2
3 // student.js
4 const students = [];
5
6 function registerStudent(studentsName) {
7   students.push({ name: studentsName });
8 }
9
10 function getStudents() {
11   return students;
12 }
13
14 export { registerStudent, getStudents };
15 export default getStudents;
16
```

import-export.js

```
1 // ESM - export and imports - how it works
2
3 // main.js
4 import * as StudentModule from './student.js';
5
6 StudentModule.registerStudent('Kate');
7 const allStudents = StudentModule.getStudents();
8
9 // main2.js
10 import fetchStudents from './student.js';
11 const allStudents = fetchStudents();
12
13 // main3.js
14 import { registerStudent } from './student.js';
15 registerStudent('Bob');
16
```



**THE
END**

A stylized title card for 'THE END'. The text is in a bold, black, sans-serif font, centered on a light blue rectangular background. This central rectangle is flanked by two pink, stylized curtain shapes that appear to be pulled back. The entire scene is enclosed within a yellow rectangular border. The background is dark and textured with wavy, organic shapes in shades of grey and black, and a few small white dots.