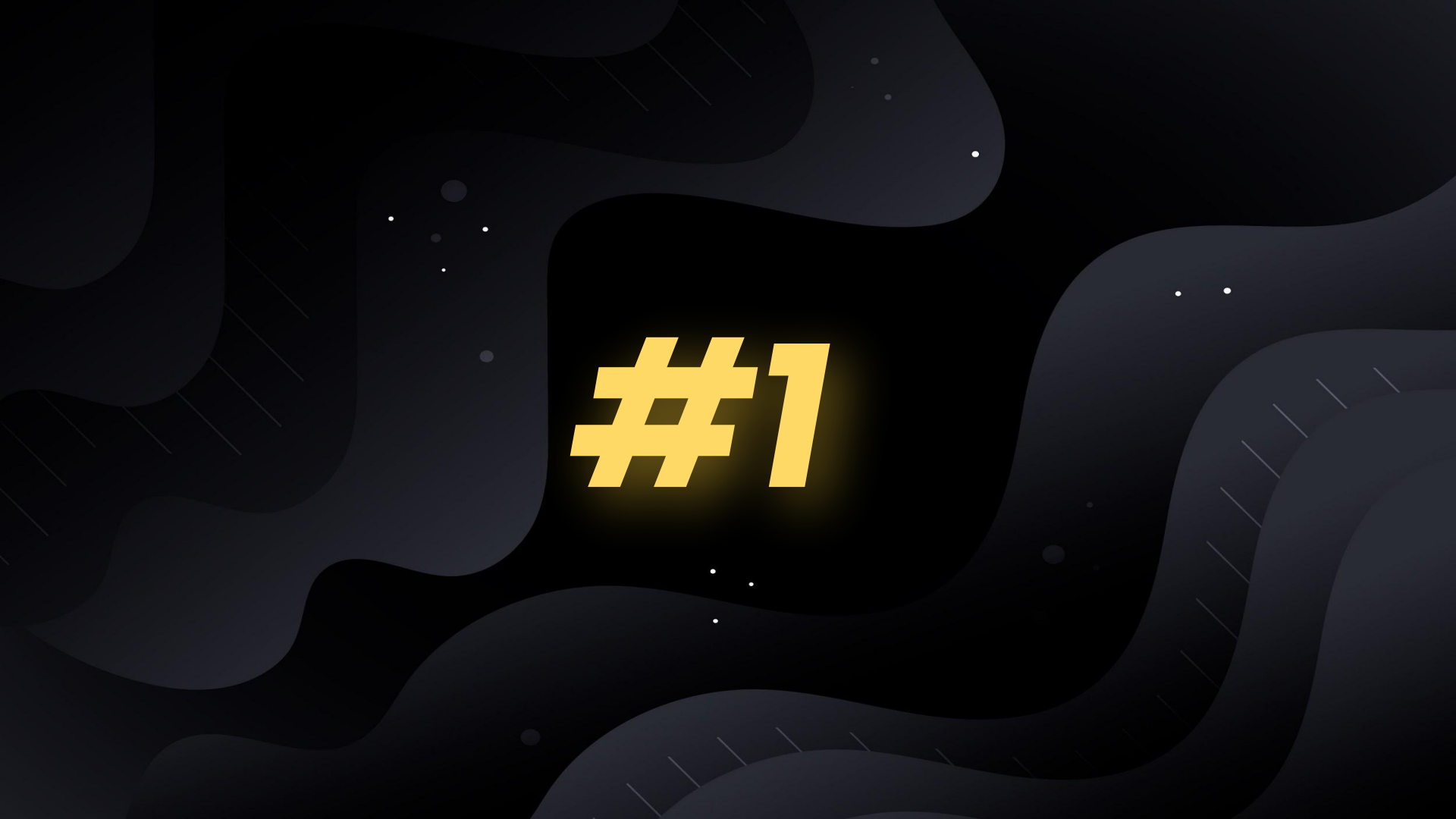
JavaScript

STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA


#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

```
// Global

const globalVariable = 'My variable';

console.log('Global scope', globalVariable); // "My variable"

if(true) {
  console.log('Block scope', globalVariable); // "My variable"
}

function logGlobalVar() {
  console.log('Function scope', globalVariable);
}

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

this.js

```
// This - defauls to global in functions context (all environments)

const transaction = {
  amount: 1000,
  updateAmount(value) {
    this.amount = value;
  },
};

function runFunctionWithTime(callback, ...args) {
  const timeStamp = Date.now();
  console.log('Running function at: ', timeStamp);
  callback(...args);
};

runFunctionWithTime(transaction.updateAmount, 2000);

console.log(transaction.amount); // 1000
console.log(globalThis.amount); // 2000 !

```

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

```
// Strict mode - this defaults to undefined in functions

'use strict';

const transaction = {
  amount: 1000,
  updateAmount(value) {
    this.amount = value;
  },
};

function runFunctionWithTime(callback, ...args) {
  const timeStamp = Date.now();
  console.log('Running function at: ', timeStamp);
  callback(...args);
}

runFunctionWithTime(transaction.updateAmount, 2000);
// Cannot set properties of undefined (setting 'amount')

```

strict.js

```
// No keyword declaration

function logArray(inputArray) {
  for (i = 0; i < inputArray.length; i++) {
    console.log(inputArray[i]);
  }
};

const hobbits = ['Frodo', 'Merry', 'Sam', 'Pippin'];
logArray(hobbits);

console.log(i); // 4 !
console.log(globalThis.i); // 4 !

```

strict.js

```
// Strict mode - no keyword declaration
'use strict';

function logArray(inputArray) {
  for (i = 0; i < inputArray.length; i++) { //ReferenceError: i is not defined
    console.log(inputArray[i]);
  }
};

const hobbits = ['Frodo', 'Merry', 'Sam', 'Pippin'];
logArray(hobbits);

console.log(globalThis.i); // undefined
console.log(i); // error

```

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

```
// Arrow functions and this keyword

const test = {
  name: 'test obj',
  logName: () => {
    console.log(this.name);
  },
};

test.logName(); // undefined
```

this.js

```
// Arrow functions and this keyword

const test = {
  name: 'test obj',
  getThisLogger() {
    return () => {
      console.log(this);
    };
  },
};

const logThis = test.getThisLogger();

logThis(); // {name: 'test obj', getThisLogger: f}

```

this.js

```
// Implicit binding

const student = {
  name: 'Bob',
  surname: 'The Builder',
  year: 1,
  logStudent() {
    console.log(`${this.name} ${this.surname} is in the ${this.year} year`);
  },
};

student.logStudent(); // Bob The Builder is in the 1 year
```

this.js

```
// New keyword

function studentCreator(name, surname) {
  this.name = name;
  this.surname = surname;
}

const student = new studentCreator('Bob', 'The Builder');
console.log(student); // {name: 'Bob', surname: 'The Builder'}

```

this.js

```
// Explicit binding

const student = {
  name: 'Bob',
  surname: 'The Builder',
};

function updateStudent(name, surname) {
  this.name = name;
  this.surname = surname;
}

console.log(student); // {name: 'Bob', surname: 'The Builder'}

updateStudent.apply(student, ['Dora', 'The Explorer']);
console.log(student); // {name: 'Dora', surname: 'The Explorer'}

updateStudent.call(student, 'Pat', 'The Postman');
console.log(student); // {name: 'Pat', surname: 'The Postman'}

```

this.js

```
// Hard binding

const studentOne = {
  name: 'Bob',
  surname: 'The Builder',
};

const studentTwo = {
  name: 'Dora',
  surname: 'The Explorer',
};

function logStudent() {
  console.log(this.name, this.surname);
}

const boundLogStudent = logStudent.bind(studentOne);

boundLogStudent(); // Bob The Builder

studentTwo.log = boundLogStudent;
studentTwo.log(); // Bob The Builder
boundLogStudent.call(studentTwo) // Bob The Builder

```

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

```
// ESM - import

import axios from 'axios'; // default import

import {map} from 'underscore'; // named import (import selected functions/variables)

import * as React from 'react'; // namespace import (import everything as)

import './initialization.js'; // side effect import

// Renaming import
import axios as apiClient from 'axios';

```

exporting

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

export.js

```
// ESM - export

// Export declaration
export const student = { name: 'Bob' };

// Export list
const studentTwo = { name: 'Kate' };
export { student, studentTwo };

// Default export
export default studentTwo;

// Aggregate export
export { studentThree, studentFour } from './exports.js'

```

import-export.js

```
// ESM - export and imports - how it works

// student.js
const students = [];

function registerStudent(studentsName) {
  students.push({ name: studentsName });
}

function getStudents() {
  return students;
}

export { registerStudent, getStudents };
export default getStudents;

```

import-export.js

```
// ESM - export and imports - how it works

// main.js
import * as StudentModule from './student.js';

StudentModule.registerStudent('Kate');
const allStudents = StudentModule.getStudents();

// main2.js
import fetchStudents from './student.js';
const allStudents = fetchStudents();

// main3.js
import { registerStudent } from './student.js';
registerStudent('Bob');

```


THE
END