

# Τεχνολογία Διαδικτύου

## 7. Javascript

Modules – Dates – Regular Expressions

Γρηγόρης Τζιάλλας

Καθηγητής

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Σχολή Θετικών Επιστημών

Πανεπιστήμιο Θεσσαλίας

# Spreading and Destructuring

# The Spread operator

- The spread syntax allows an array or a string to expand in the places where arguments or elements are expected.
- Whenever you use ...arr in the function call, it expands an iterable object arr to the list of arguments.

```
let arr = [6, 8, 2];  
let max = Math.max(...arr);  
console.log(max); // 8
```

```
//combining spread operators  
let arr1 = [2, -3, 5, 4];  
let arr2 = [9, 2, -7, 2];  
let max1 = Math.max(...arr1, 56, ...arr2, -9);  
console.log(max1); // 56
```

```
//spreading a string  
let str = "string";  
console.log(...str); // s,t,r,i,n,g
```

# The Spread operator for objects

- Spread syntax helps us clone an object with the most straightforward syntax using the curly braces and three dots {...}.
- With spread syntax we can clone, update, and merge objects in an immutable way.

```
//spreading an object
let nick = { name: "Nick", age: 22 };
//cloning nick to petros
let peter = {...nick};
peter.name = "peter";
console.log(nick, peter)
;

//merging two objects into another object
let student = { grade: 9 };
let nickFromSpread = {...nick, ...student};
console.log(nickFromSpread);
```

# Destructuring

- The destructuring assignment is a unique syntax that helps “to unpack” objects or arrays into a group of variables. Destructuring can also operate efficiently with complex functions, default values, and more.
- The destructuring assignment uses the following syntax:

```
const anArray = [1, 2, 3, 4, 5];  
const [m,n] = anArray;  
console.log(m); // 1  
console.log(n); // 2
```

# Destructuring ...rest

- In case it is necessary to get not only the first values but also to assemble all the followings, you have the option of adding another parameter for getting “the rest” by just using three dots “...”, like this:

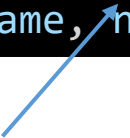
```
let [name, surname, ...rest] = ["John", "Doe", "doctor", "surgeon"];
console.log(name); // John
console.log(surname); // Doe
// Note that type of `rest` is Array.
console.log(rest[0]); // doctor
console.log(rest[1]); // surgeon
console.log(rest.length); // 2
```

# Destructuring objects

- You can use the destructuring assignment with objects, as well, by using the following basic syntax:

```
let { var1, var2 } = { var1: ..., var2: ... }
```

```
//destructuring an object  
let nick = { name: "Nick", age: 22 };  
let { name, age: nickAge } = nick;  
console.log(name, nickAge);
```



For assigning a property to a variable with different name, then you can act like this:

# Modules



# Modules

- JavaScript modules allow you to break up your code into separate files. This makes it easier to maintain the code-base.
- JavaScript modules rely on the import and export statements.
- You can export a function or variable from any file.
- There are two types of exports: Named and Default.
- Named exports
  - You can create named exports two ways. In-line individually, or all at once at the bottom.
- Default exports
  - You can only have one default export in a file.

# Examples of exports

## Inline named exports

```
//inline named export
export const name = "Nick";
export const age = 18;
export const message = () => {
  return name + ' is ' + age +
    'years old.';
};
```

## Named exports at the bottom

```
const name = "Nick";
const age = 18;

//named export at the bottom
export {name, age};
```

## Importing named exports

```
//importing a named export
import {name, age} from "./myModule.js";
```

## Default export

```
//Class definition
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  //property
  height = undefined;
  //method
  talk() {
    console.log("I am " + this.name +
      " and my age is " + this.age);
  }
}


export default Person;
```

## Importing default export

```
//importing a default export
import Person from "./Person.js"
```

# Nested modules

```
import { Student } from "./Student.js";  
let aris = new Student("Aris", 19,  
    "Τμήμα Πληροφορικής");  
aris.grade = 7.5;  
console.log(aris);
```



```
//Class definition  
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
    //property  
    height = undefined;  
    //method  
    talk() {  
        console.log("I am " + this.name +  
            " and my age is " + this.age);  
    }  
}  
  
export default Person;
```

```
//importing a default export  
import Person from "./Person.js"  
//Definition of a subclass of Student  
Export class Student extends Person {  
    constructor(name, age, school) {  
        //initialize superclass  
        super(name, age);  
        this.school = school;  
    }  
    //private property  
    #grade = undefined;  
    //Getter settter for grade  
    get grade() {  
        //report that someone accessed the grade  
        console.log("returning grade " + this.#grade);  
        return this.#grade;  
    }  
    set grade(x) {  
        //check grade  
        if (x >= 0 && (x <= 10)) {  
            console.log("changing grade to " + x);  
            this.#grade = x;  
        }  
        else {  
            //throw error if invalid grade  
            throw ("Invalid grade " + x);  
        }  
    }  
}
```

# Dates – Regular Expressions

# The Date Object

- JavaScript Stores Dates as Milliseconds
  - JavaScript stores dates as number of milliseconds since January 01, 1970.
- Date objects are created with the new Date() constructor.
  - new Date()
  - new Date(date string)
  - new Date(year,month)
  - new Date(year,month,day)
  - new Date(year,month,day,hours)
  - new Date(year,month,day,hours,minutes)
  - new Date(year,month,day,hours,minutes,seconds)
  - new Date(year,month,day,hours,minutes,seconds,ms)
  - new Date(milliseconds)

# Date Methods

- `toString()` converts a date to a readable format
- `toLocaleString` returns a date as a string, using locale settings
- `toUTCString()` converts a date to a string using the UTC standard
- `parse()` returns the number of milliseconds from 1/1/1970

Method	Description
<code>getFullYear()</code>	Get <b>year</b> as a four digit number (yyyy)
<code>getMonth()</code>	Get <b>month</b> as a number (0-11)
<code>getDate()</code>	Get <b>day</b> as a number (1-31)
<code>getDay()</code>	Get <b>weekday</b> as a number (0-6)
<code>getHours()</code>	Get <b>hour</b> (0-23)
<code>getMinutes()</code>	Get <b>minute</b> (0-59)
<code>getSeconds()</code>	Get <b>second</b> (0-59)
<code>getMilliseconds()</code>	Get <b>millisecond</b> (0-999)
<code>getTime()</code>	Get <b>time</b> (milliseconds since January 1, 1970)

[Complete JavaScript Date Reference](#)

# Regular expressions

- A regular expression is a sequence of characters that forms a search pattern.

- Syntax

*/pattern/modifiers;*

*Modifiers can be used to perform case-insensitive, global and multiline searches:*

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

# Regular expressions

- A regular expression is a sequence of characters that forms a search pattern.

- Syntax

*/pattern/modifiers;*

*Modifiers can be used to perform case-insensitive more global searches:*

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching



# Regular Expression patterns

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x y)	Find any of the alternatives separated with

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifier	Description
n+	Matches any string that contains at least one <i>n</i>
n*	Matches any string that contains zero or more occurrences of <i>n</i>
n?	Matches any string that contains zero or one occurrences of <i>n</i>

# Regex Example

```
let text = "Welcome to www";
let n = text.search(/com/i);
console.log(n); //3

let m = text.match(/e/g);
console.log(m);

let text1 = "123456789";
let result1 = text1.match(/[1-4]/g);
console.log(result1);

let text2 = "Hellooo ooo oo";
let result2 = text2.match(/o+/g);
console.log(result2);

let text3 = "123.231";
let result3 = text3.match(/[0-9]+/g);
console.log(result3);
```

3

▶ (2) ['e', 'e']

▶ (4) ['1', '2', '3', '4']

▶ (3) ['ooo', 'ooo', 'oo']

▶ (2) ['123', '231']

# Various Operators

# Javascript handling of null / undefined

- Nullish coalescing operator (??)
  - `val1 ?? val2`; - Returns `val2` if `val1` is null or undefined otherwise `val1`
    - like `val1 || val2`
  - `param ?? 32`; - Works for all number values of `param` including 0
- Optional Chaining (?.)
  - `obj?.prop` - Returns undefined if `obj` is undefined or null otherwise returns `obj.prop`
  - `obj?.subobj?.prop`; - Handles `obj` or `subobj` being null or undefined.
  - `func?.()`; - Calls `func` only if not null or undefined.
  - `arr?.[1]` - Access array only if `arr` is not undefined or null.
- BigInt - Bigger than 53bit integers - Example: `9007199254740992n`
  - Makes working with 64bit integers from other languages easier

# Some JavaScript idioms

- Assign a default value
  - `hostname = hostname || "localhost";`
  - `port = port || 80;`
- Access a possibly undefined object property
  - `let prop = obj && obj.propname;`
- Handling multiple this by using another variable (e.g. self)

```
let self = this;
```

```
fs.readFile(self.fileName + fileNo, function (err, data) {  
    console.log(self.fileName,fileNo);  
});
```

# Αναφορές

- <https://developer.mozilla.org/en-US/docs/Learn/JavaScript>
- <https://www.w3schools.com/js/default.asp>
- [https://www.w3schools.com/js/js\\_examples.asp](https://www.w3schools.com/js/js_examples.asp)
- <https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/#basic-javascript>