



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

JavaScript Parte III

IIC2513 - Tecnologías y Aplicaciones Web

Sebastián Vicencio R.
2do Semestre 2020

Operaciones útiles

JavaScript moderno

Operaciones útiles

Rest parameters

Spread syntax
`...myVariable`

Podemos utilizarlo con arrays en tres situaciones

1. En parámetros de una función
2. Al llamar a una función
3. Crear nuevos arrays

Operaciones útiles

Rest parameters - Parámetros función

Spread syntax
`...myVariable`

```
function getRestParams(firstArg, ...rest) {  
    console.log(rest);  
}
```

```
getRestParams(1, 'test', true, 20);
```

Operaciones útiles

Rest parameters - Llamado funciones

Spread syntax
`...myVariable`

```
const numbers = [1, 4, 5, 2];
```

```
Math.max(...numbers);
```

Operaciones útiles

Rest parameters - Nuevos arrays

Spread syntax
`...myVariable`

```
// Create new arrays  
const clonedArray = [...numbers];  
  
const newArray = [...numbers, 10, 13];
```

Operaciones útiles

Destructuring

Spread syntax
`...myVariable`

Forma de manejar datos de un objeto o array

1. Asignar a variables
2. Crear nuevos objetos

Operaciones útiles

Destructuring

Spread syntax `...myVariable`

```
const numbers = [1, 4, 5, 2];  
const person = { name: 'Student', age: 22 };  
  
const [firstNumber, secondNumber, ...restNumbers] = numbers;  
  
const { name, age } = person;  
  
// Create new object  
const newObj = { ...person };
```


Prototype

Y herencia en JavaScript

Prototypes

Preguntémonos lo siguiente

```
const newArray = [];
```

```
newArray.length
```

```
newArray.map
```

```
newArray.reduce
```

¿Cómo un array **obtiene propiedades y métodos?**

Prototypes

Recordemos ahora los constructores

```
function Person(name, age, city, isStudent = false) {  
  this.name = name;  
  this.age = age;  
  this.city = city;  
  this.isStudent = isStudent;  
  this.talk = function() {  
    console.log('My name is ' + this.name + '. I live in ' + this.city);  
  }  
}
```

Prototypes

Recordemos ahora los constructores

```
const person1 = new Person('John', 26, 'New York');
```

```
const person2 = new Person('Helen', 32, 'Berlin');
```

```
person1.talk === person2.talk;
```

¿Qué pasa con esta **comparación**?

Prototypes

Definición

JavaScript es un **lenguaje** basado en **prototipos**

Todo objeto **tiene un objeto** llamado **prototype**

Hereda propiedades y métodos de este prototype

Prototypes

Prototype chain



El prototype **también tiene su propio prototype**

Eslabón final es Object, cuyo prototype es null

Prototypes

Prototype chain

¿Cómo se accede al prototipo?

```
person1.toString();
```

```
person1.__proto__;
```

```
Object.getPrototypeOf(person1);
```

Prototypes

Propiedad prototype

Propiedades y métodos heredables en objeto
bajo key prototype

Person.prototype

// And one of the base object

Object.prototype

Array.prototype

Prototypes

Función constructora - Propiedad constructor

Toda **función constructora** tiene en su prototype la **key constructor**

```
person1.constructor
```

```
person1.constructor.name
```

Prototypes

Modificación prototype

Agregamos propiedad a objeto prototype

```
Person.prototype.introduce = function() {  
    console.log(`I am ${this.isStudent ? '' : 'not '}a student`)  
}
```

```
person1.introduce();
```

Prototypes

Modificación prototype

En general se suelen definir **propiedades en constructor y métodos en prototype**

```
function OtherPerson(name) {  
    this.name = name;  
}
```

```
OtherPerson.prototype.talk = function() {  
    console.log('My name is ' + this.name);  
}
```

Referencias

- MDN - Spread syntax (...)
- MDN - Object prototypes
- Eloquent JavaScript - The Secret Life of Objects