UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

Tecnologias e Programação de Sistemas de Informação

JS PROTOTYPE & NODE EVENTS

Desenvolvimento Web - Back-End | David Jardim

Cofinanciado por:

MADEIRA 14-20

REGIÃO AUTÓNOMA DA MADEIRA

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Social Europeu

JSON:
"JAVASCRIPT OBJECT NOTATION" – A STANDARD FOR STRUCTURING DATA THAT IS INSPIRED BY JAVASCRIPT OBJECT LITERALS

Javascript engines are built to understand it.

```
{
   "firstname": "John",
   "lastname": "Doe",
   "address": {
      "street": "101 Main St.",
      "city": "New York",
      "state": "NY"
   }
}
```

Why use JSON?

- Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:
  - JSON.parse()

- So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

Object creation in JS via constructor function syntax

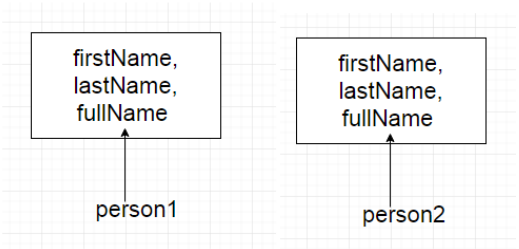
```
function Person(firstName, lastName) {
    this.firstName = firstName,
    this.lastName = lastName,
    this.fullName = function() {
        return this.firstName + " " + this.lastName;
    }
}
```

Object creation in JS via constructor function syntax

```
var john = new Person("John", "Doe");
var jane = new Person("Jane", "Doe");
```

- Two copies of the constructor function was created for each person
- We have **two** instances of function *fullName* that do the same thing

- How can we solve this?



Prototypes in Javascript

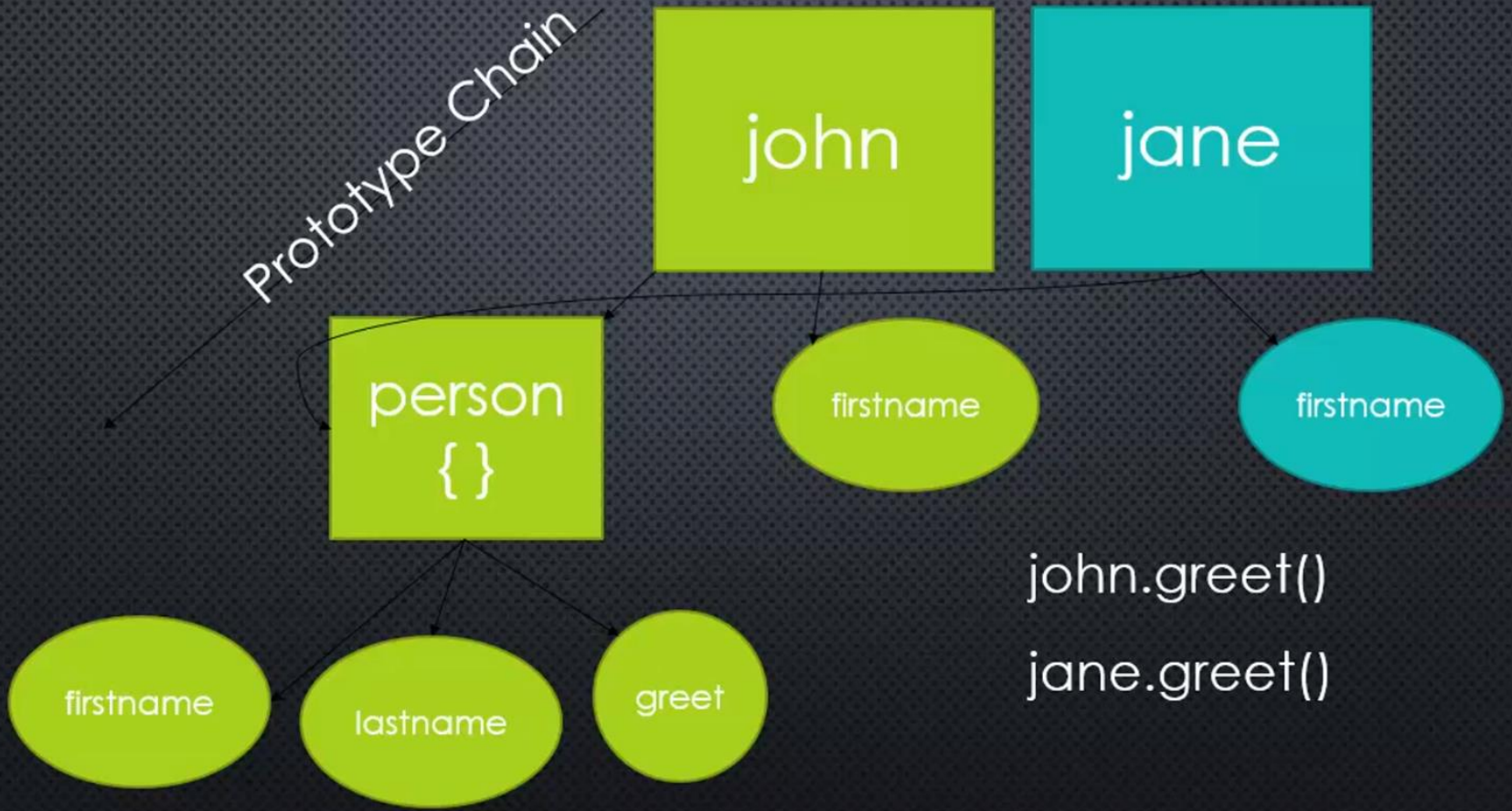
- Upon creation of a function in JavaScript, JavaScript engine adds a **prototype property** to the function
- This prototype property is an **object** which has a constructor property by default
- We can access the function's prototype property using the syntax ***functionName*.prototype**
- **TASK:** DEBUG and search for the *__proto__* property

Prototypal inheritance in JS

```
function Person(firstName, lastName) {
    this.firstName = firstName,
    this.lastName = lastName
}

Person.prototype.greet = function() {
    console.log(" Hello" + this.firstName + " " + this.lastName);
}

var john = new Person("John", "Doe");
john.greet();

var jane = new Person("Jane", "Doe");
jane.greet();

console.log(john.__proto__);
console.log(jane.__proto__);
console.log(john.__proto__ == jane.__proto__);
```

Prototypal inheritance in JS

- As prototype is an object, we can attach **properties** and **methods** to the prototype
- All the objects created using the constructor function will share those properties and methods
- An object's prototype may also have a prototype object, which it inherits methods and properties from. This is often referred to as a **prototype chain.**


Prototype Chain
john
jane
person
{ }
firstname
firstname
firstname
lastname
greet
john.greet()
jane.greet()

Object creation in JS via Object.Create and prototypes

```
var person = {
    firstname: '',
    lastname: '',
    greet: function() {
        return this.firstname + ' ' + this.lastname;
    }
}
```

Object creation in JS via Object.Create and prototypes

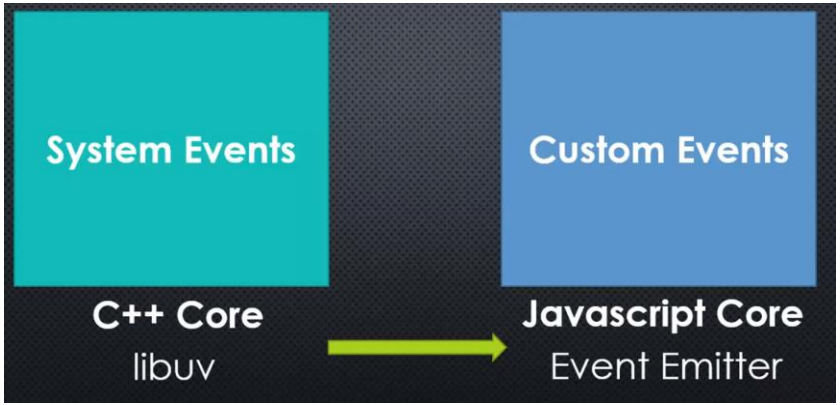
```
 9 var john = Object.create(person);
10 john.firstname = 'John';
11 john.lastname = 'Doe';
12
13 var jane = Object.create(person);
14 jane.firstname = 'Jane';
15 jane.lastname = 'Doe';
```

Object creation via Class Declaration

```
class Person {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = 0;
        this.greet = function () {
            return this.firstName + " " + this.lastName;
        };
    }
}
```

Events in Node?

- System Events – C++ Core
  - Each time a peer connects to a server
  - Each time a file is opened
  - When a stream is ready to be read
- Custom Events – Javascript Core
  - Event emitter
  - Create our own events



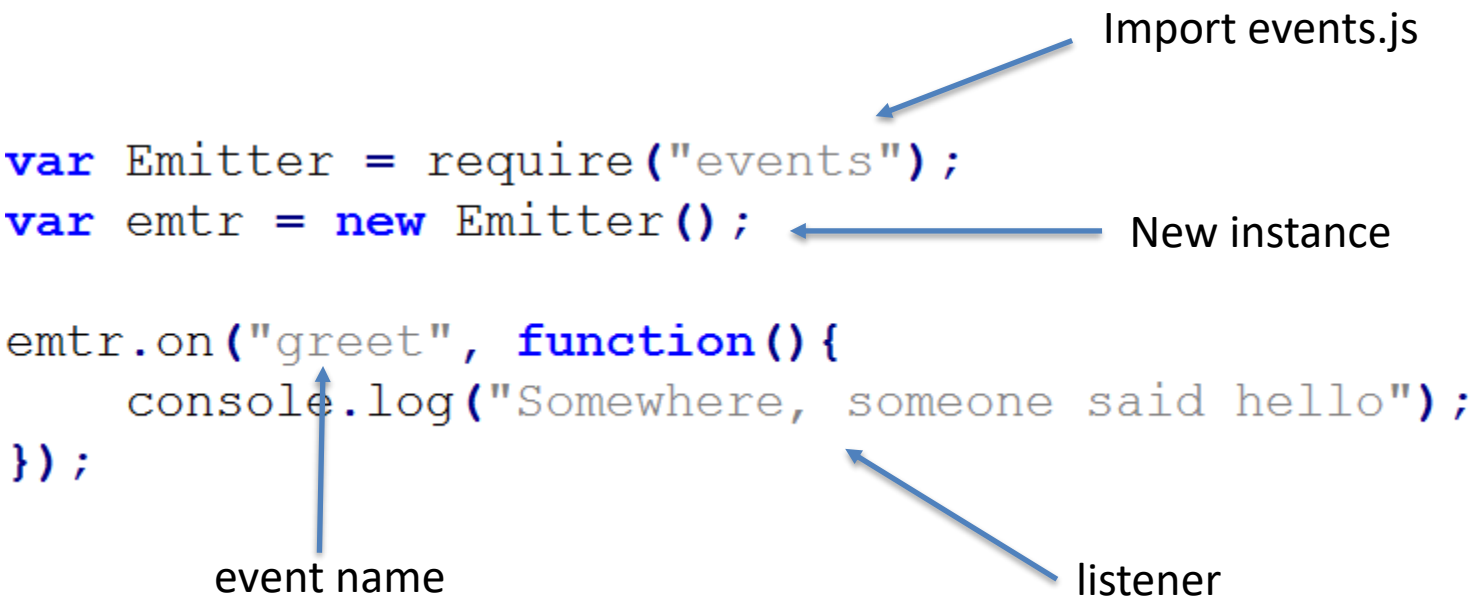
EVENT:
SOMETHING THAT HAS HAPPENED IN OUR APP THAT WE CAN RESPOND TO.

In Node we actually talk about two different kinds of events.

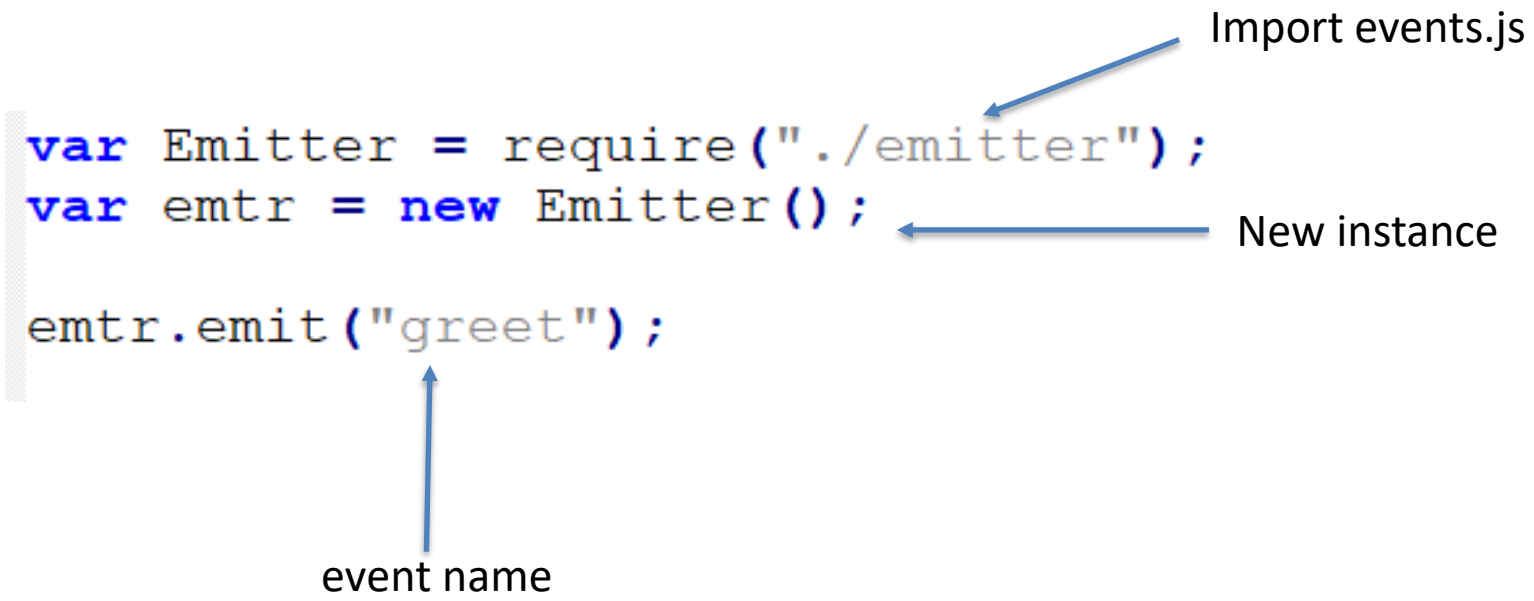
EVENT LISTENER: THE CODE THAT RESPONDS TO AN EVENT.

In Javascript's case, the listener will be a function.

Register an event with an Event Emitter in Node



Emit an event with an Event Emitter in Node



MAGIC STRING: A STRING THAT HAS SOME SPECIAL MEANING IN OUR CODE.

This is bad because it makes it easy for a typo to cause a bug, and hard for tools to help us find it.

Solution? Create a file to store all the events names

```
module.exports = {
    events:{
        GREET: 'greet',
        FILESAVED: 'filesaved',
        FILEOPENED: 'fileopened',
    }
}
```

event name

property name

object

```
var eventConstants = require('./config');
```


UNIVERSIDADE da MADEIRA
CTeSP
CURSOS TÉCNICOS
SUPERIORES PROFISSIONAIS