



JavaScript for Enterprise Development

Week 2-1



CONTENTS

JS: Prototype
Inheritance

JS: Classes

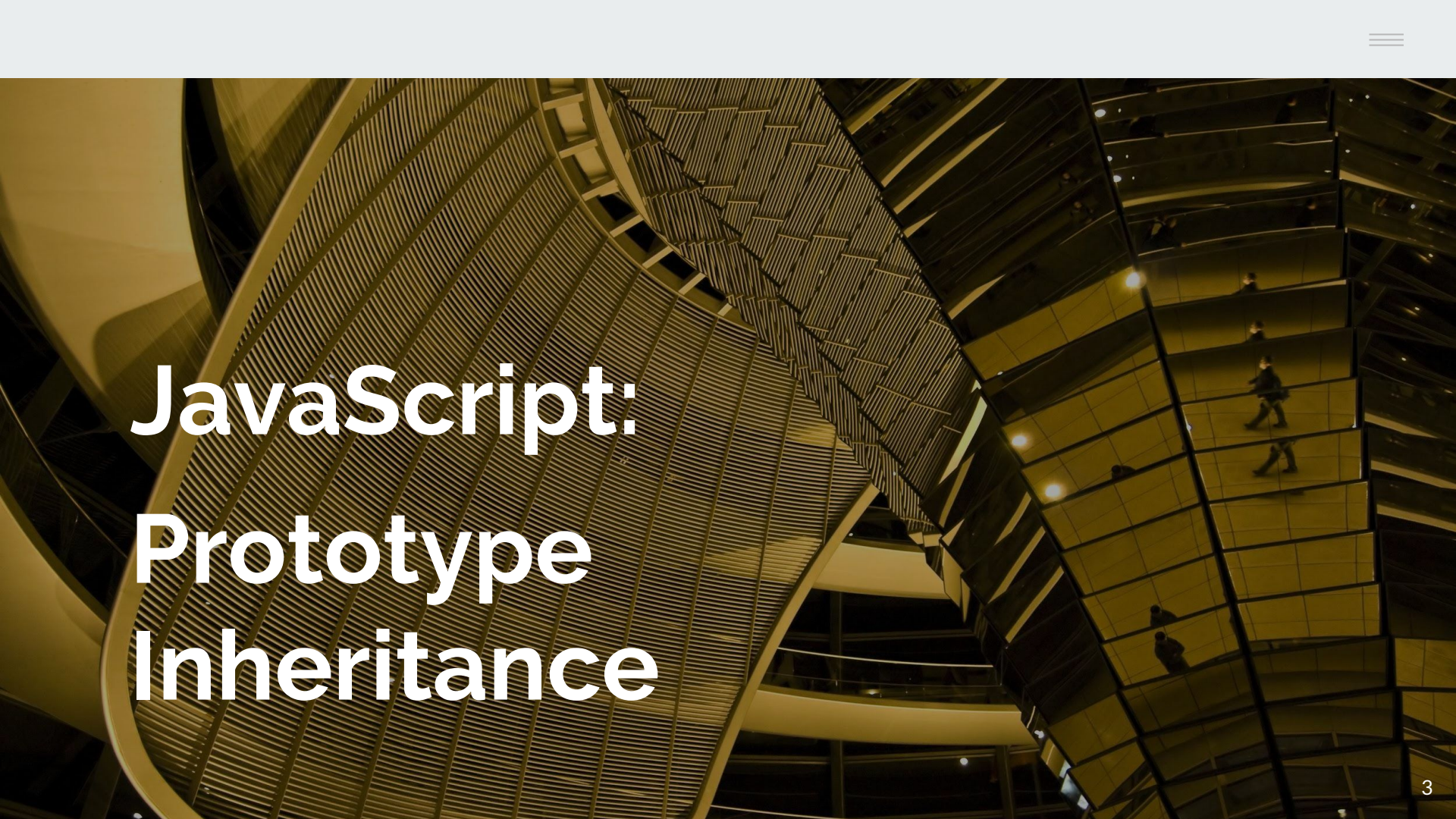
JS: Modules

Package
Management Tools

Package Versioning

Build Tools & CI

Assignment



JavaScript: Prototype Inheritance

Objects and Context



In a function definition, `this` refers to the "owner" of the function.

```
var person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};
```


Objects and Context



- When used alone, `this` refers to the Global object (*window*).
- When used in a function, `this` also refers to the Global object (In strict mode, `this` will be *undefined*)
- In object definition context (earlier examples), `this` is the “owner” object reference

The `call()` and `apply()` methods can be used to call an object method with another object as argument.

Objects and Context



```
var person1 = {  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
};  
  
var person2 = {  
  firstName: "John",  
  lastName: "Doe",  
};  
  
person1.fullName.call(person2); // Will return "John Doe"
```

Constructors



Sometimes we need a "blueprint" for creating many objects of the same "type".

The way to create an "object type", is to use an object **constructor** function with a **new** keyword.

```
function Person(first, last, age) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
}  
  
var myFather = new Person("John", "Doe", 50);  
var myMother = new Person("Sally", "Rally", 48);
```

Execution context

is the value of `this` which is a reference to the object that owns the current executing code.

We recommend additional readings about edge cases of context creation

Job interview most popular questions

Inheritance



Though JavaScript have Objects, there is no classes in “vanilla” JavaScript. But nevertheless it is possible to adopt object-oriented style with JavaScript with help of **prototype inheritance**.

All JavaScript objects inherit properties and methods from a **prototype**.

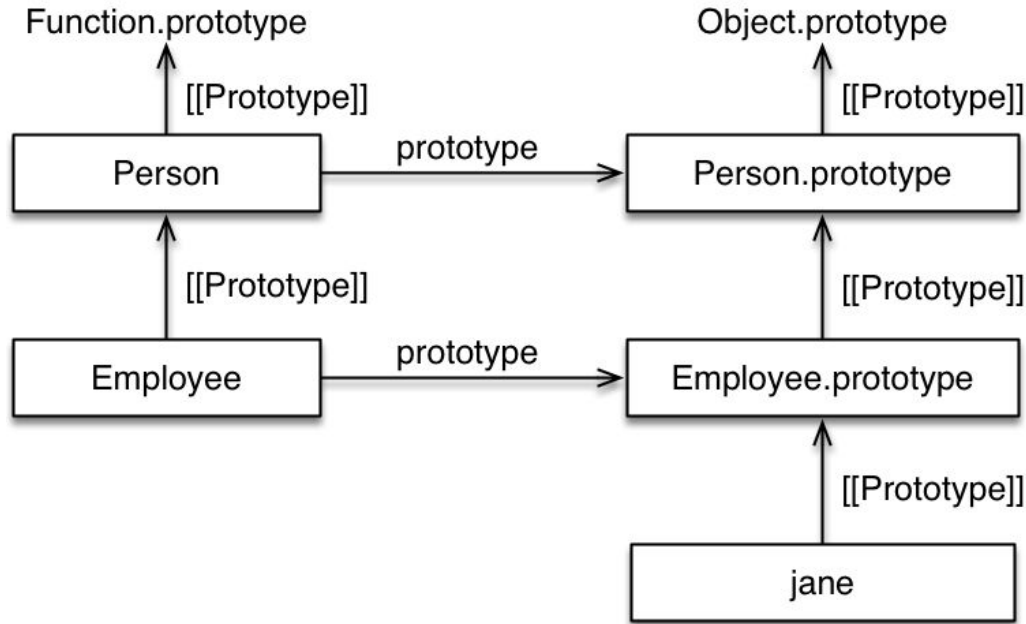
Date objects inherit from **Date.prototype**. Array objects inherit from **Array.prototype**. Person objects inherit from **Person.prototype**.

The **Object.prototype** is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from **Object.prototype**.



Inheritance



Prototype inheritance is widely used in large JavaScript application (at least under the hood of modern toolkits)

Job interview most popular questions



JavaScript: Classes



Classes in JavaScript

JavaScript classes, introduced in ECMAScript 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance.

The class syntax does not introduce a new object-oriented inheritance model to JavaScript.



Class declarations

One way to define a class is using a class declaration. To declare a class, you use the class keyword with the name of the class (**constructor** is a special method for initializing an object created with a class):

```
class Rectangle {  
    constructor(height, width) {  
        this.height = height;  
        this.width = width;  
    }  
}
```



Class methods

```
class Rectangle {  
  constructor(height, width) {  
    this.height = height;  
    this.width = width;  
  }  
  
  calcArea() { // method  
    return this.height * this.width;  
  }  
}  
  
const square = new Rectangle(10, 10);  
console.log(square.calcArea());
```




Class getters

```
class Rectangle {  
  
    ...  
  
    get area() { // getter  
        return this.calcArea();  
    }  
}  
  
const square = new Rectangle(10, 10);  
console.log(square.area);
```



Class static methods

```
class Rectangle {  
    ...  
  
    static isSquare(height, width) {  
        return height === width; // no access to `this`  
    }  
}
```



Class fields (proposal)

```
class Car {  
  hasWheels = true;  
  color;  
  static driveable = true;  
}
```



Subclasses

```
class Animal {  
    constructor(name) {  
        this.name = name;  
    }  
  
    speak() {  
        alert(this.name + ' squeaks');  
    }  
}
```

```
class Dog extends Animal {  
    constructor(name) {  
        super(name);  
    }  
  
    speak() {  
        alert(this.name + ' barks.');
```

```
(new Dog(Bobik')).speak();// Bobik barks.
```



Decorators (draft)

```
@honkable
@driveable
class Car {
    hasWheels = true;
    color;
}
```



JavaScript: Modules



AMD: Asynchronous module definition

The oldest specification. Could run in non-transpiled environment. Currently obsolete.

Examples: RequireJS, Dojo



```
requirejs.config({  
  paths: { app: 'src' }  
});
```

// index.js

// Requires src/main.js file

```
requirejs(['app/main'], function(main) {  
  main(); // Use exported `main` variable here  
});
```

```
define(function (require) {  
  return function() {  
    alert('Hello!');  
  }  
});
```

// src/main.js



CommonJS: Hello from Node.js

Synchronous module import specification used primarily in node.js environment

```
var main = require('./src/main');
```

```
main();
```

// index.js

```
var msg = function() {  
    alert('Hello!');  
}
```

```
module.exports = msg;
```

// src/main.js



ES6 modules: Part of modern language specs

“EcmaScript 6 modules” is part of modern JavaScript standards specifications widely used in pre-transpiled client-side JS



```
import { main } from './src/main';
```

```
// index.js
```

```
main();
```

```
export const main = () => {
```

```
  alert('Hello!');
```

```
}
```

```
// src/main.js
```



```
import main from './src/main';
```

// index.js

```
main();
```

```
const msg = () => {  
  alert('Hello!');  
}
```

// src/main.js

```
export default msg;
```





Module requesting strategies

- Request and cache each module via HTTP in runtime - dramatically reduces initial app size and loading time, application became unresponsive
- Statically analyze all imports in codebase and concatenate modules into one big bundle - single HTTP request on application startup, costs increased initial app size and loading time
- Hybrid - split your application into separate chunks, each containing bundled together modules



Package management tools

Package managers evolution



NPM
2010



Yarn
2016



NPM 5
2017

Dark era of web
development

Bower
2012



Nowadays



npm

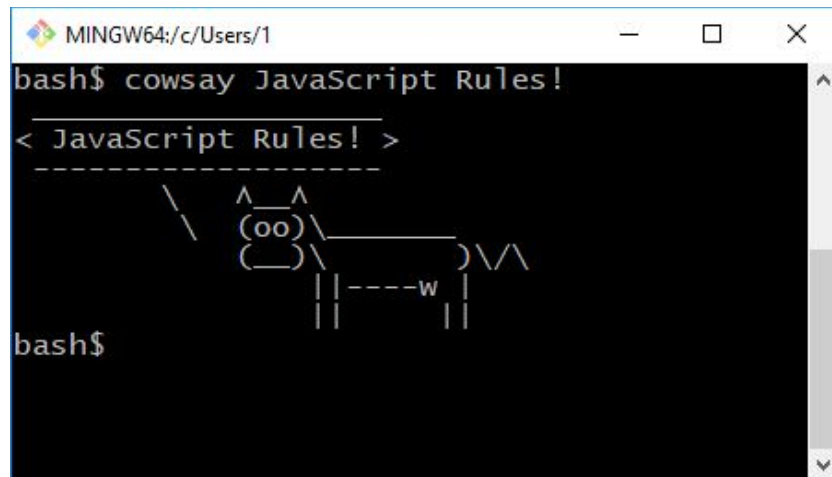
npm is a command line tool to help you manage packages from global npm registry (npmjs.com)

npm could install those packages on your machine:

- globally - they will be available all over your system
- or locally - available only in the context of your project (or directory)

npm global packages

```
npm install -g cowsay
```



A terminal window titled 'MINGW64:/c/Users/1' showing the command 'cowsay JavaScript Rules!' being executed. The output is a ASCII art cow saying 'JavaScript Rules!'. The prompt 'bash\$' is visible at the bottom left of the terminal.

```
bash$ cowsay JavaScript Rules!  
< JavaScript Rules! >  
-----  
      \      ^__^  
       (oo)\_____  
          (__)\\       )\\/\  
              ||----w |  
              ||     ||  
bash$
```

npm global packages



```
npm install cowsay
```

```
npm WARN: no such file or directory, open 'my-project\package.json'
```

```
npm notice created a lockfile as package-lock.json.
```

```
You should commit this file.
```

```
npm WARN: no such file or directory, open 'my-project\package.json'
```

```
npm WARN my-project No description
```

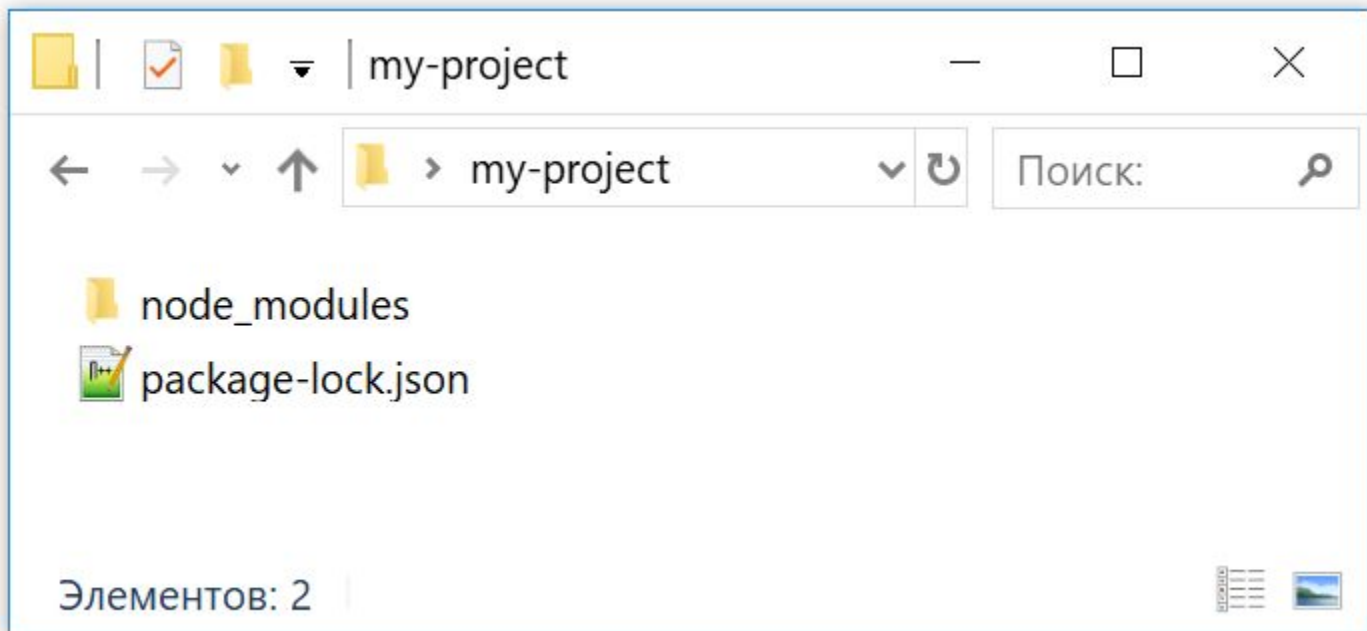
```
npm WARN my-project No repository field.
```

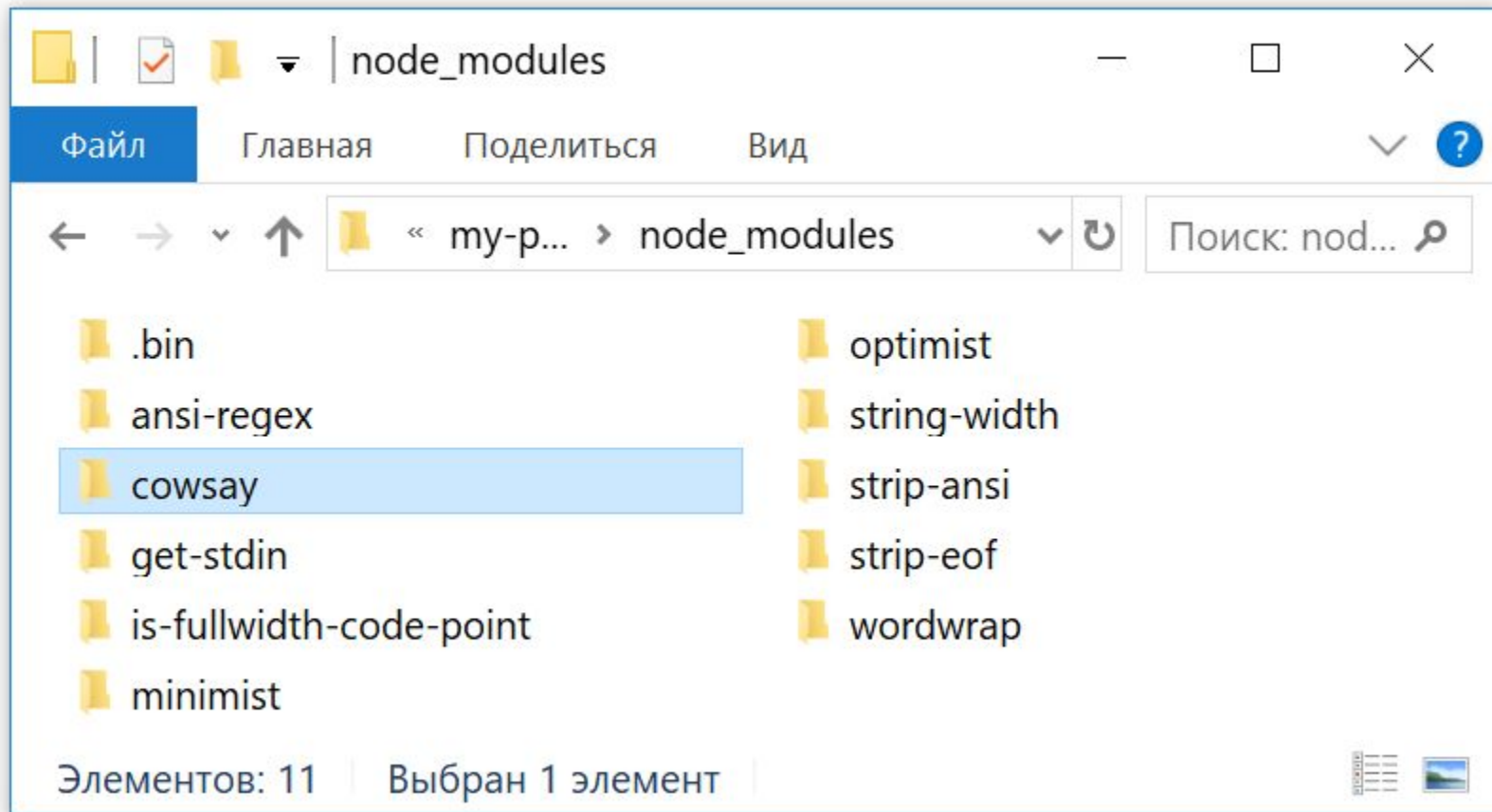
```
npm WARN my-project No README data
```

```
npm WARN my-project No license field.
```

```
+ cowsay@1.3.1
```

```
added 10 packages in 1.146s
```







package.json

Is used to manage locally installed npm packages.

package.json file:

- lists the packages that your project depends on.
- allows you to specify the versions of a package that your project can use using semantic versioning rules.
- makes your build reproducible, and therefore much easier to share with other developers.



minimal package.json

```
{  
  "name": "my_awesome_project",  
  "description": "Data visualization and analysis web client",  
  "version": "1.0.0",  
  "devDependencies": {  
    "cowsay": "1.3.1"  
  }  
}
```



Installing project dependencies

With `package.json` in your project directory you could just run

```
npm install
```

or even

```
npm i
```

Package versioning



Semantic versioning

Every project in npm registry have one or more versions.

Version numbers are conforms semantic versioning. Examples:

```
react@16.3.2
```

```
lodash@0.5.0-rc.1
```

```
my-awesome-lib@1.0.0-beta
```



What does each version number means

Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.
- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.



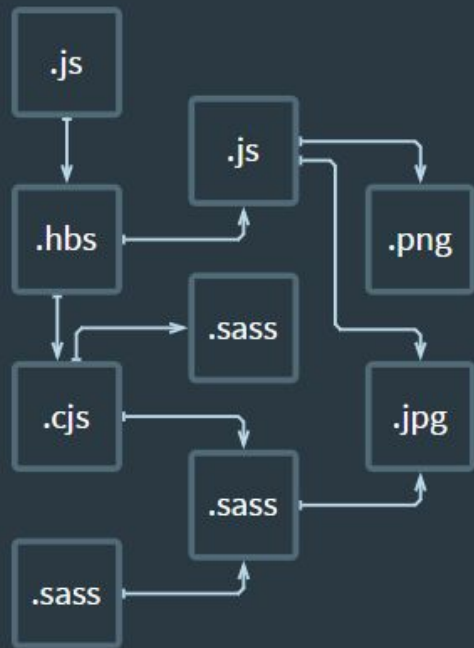
Version ranges

As a package “consumer”, you can specify which kinds of updates your app can accept in the `package.json` file:

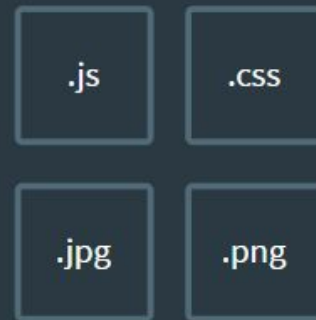
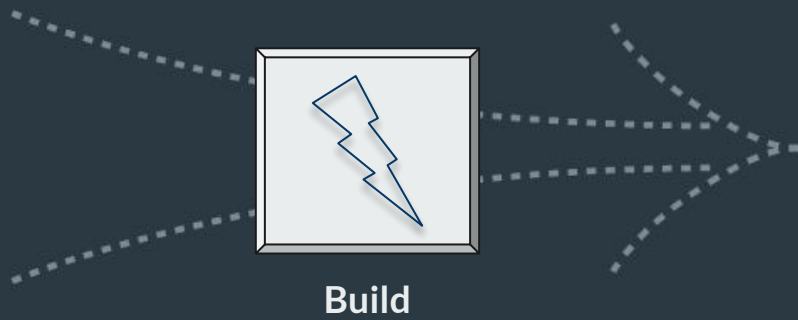
- Patch releases: 1.0 or 1.0.x or ~1.0.4
- Minor releases: 1 or 1.x or ^1.0.4
- Major releases: * or x



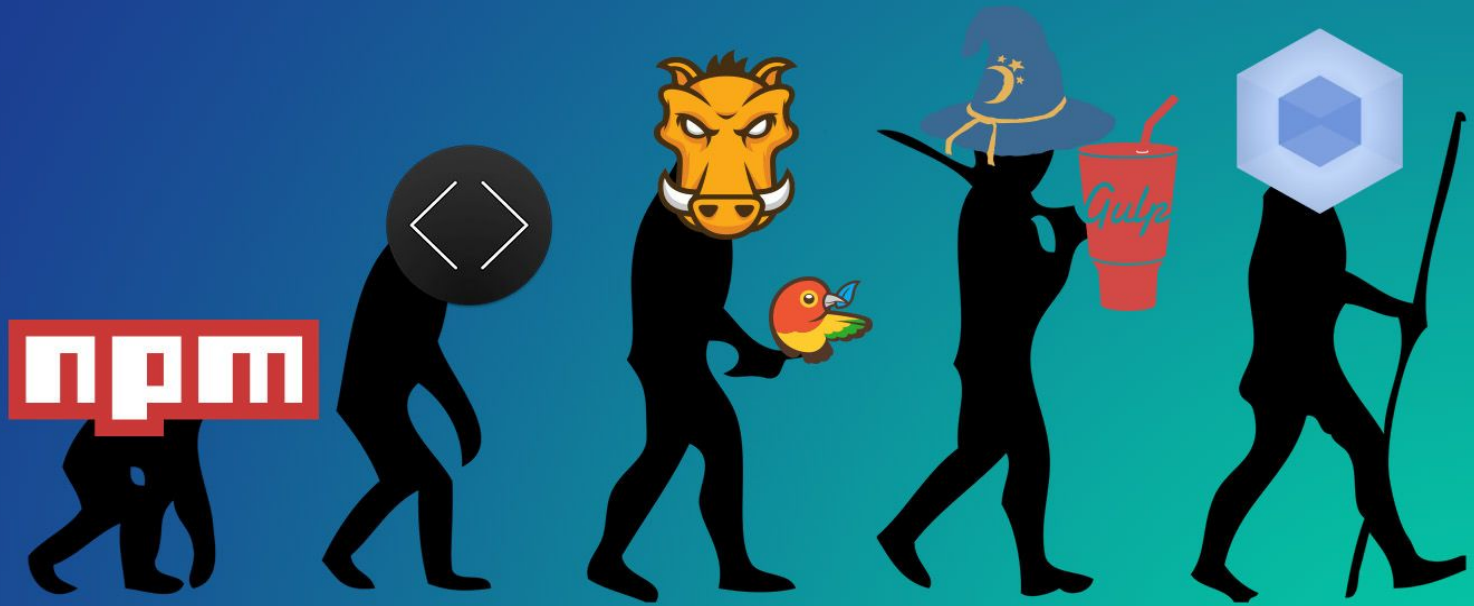
Build tools



MODULES WITH DEPENDENCIES



STATIC ASSETS





Build tools types

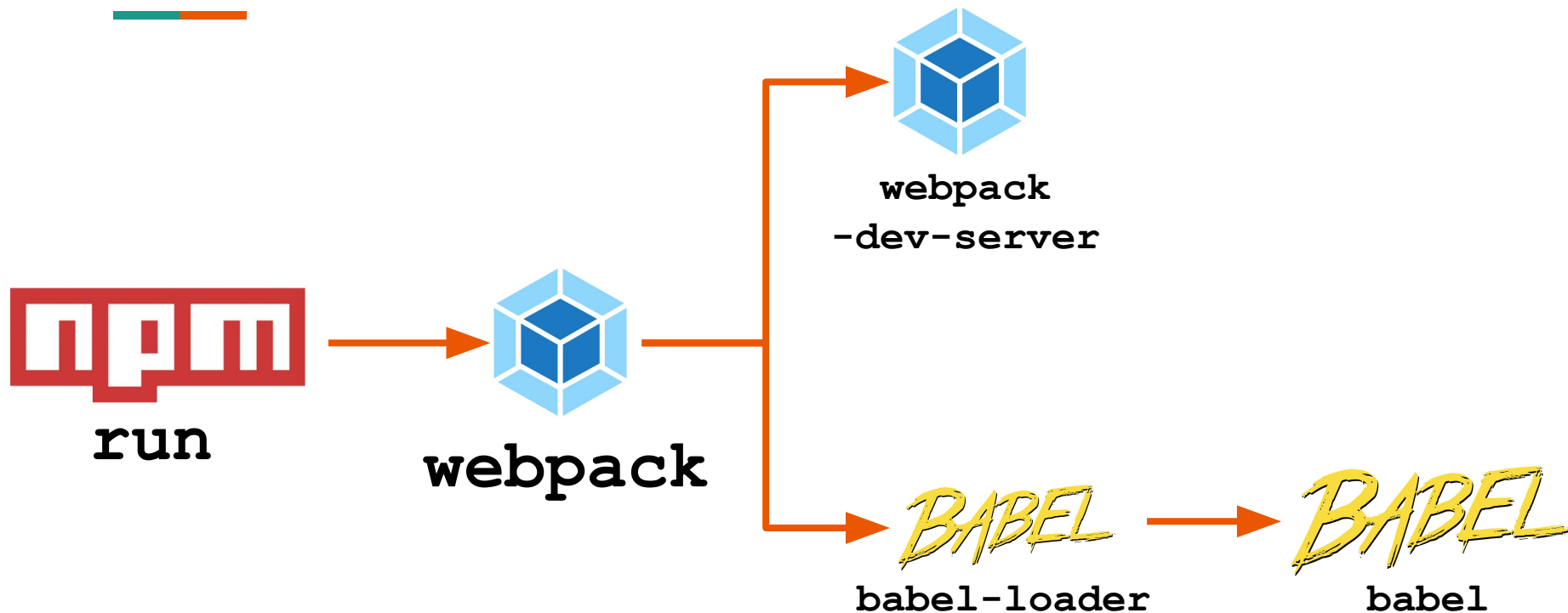
Task runners: npm, Grunt

Bundlers: Webpack, Parcel

Transpilers: Babel presets

Compilers: TypeScript, React JSX

Classic toolchain



Assignment



Assignment for the next week

- Add npm support to your project (`package.json`).
- Configure development environment for your application:
 - Configure project bundler (webpack / parcel / etc)
 - Set up development web server (webpack-dev-server / smth else)
 - Add code transpiling support
- Check for ES-imports support
- Make a PR



Links

Node.js download page (8.11 LTS recommended): <https://nodejs.org/en/download/>

How to set up minimal build toolchain: <http://ccoenraets.github.io/es6-tutorial/setup-webpack/>

Webpack-dev-server: <https://github.com/webpack/docs/wiki/webpack-dev-server>

Common development guide on webpack: <https://webpack.js.org/guides/development/>



That's all!





Resources

- Lectures, resources and course project requirements:
<https://github.com/kos33rd/web-developer-course>
- Our telegram group:
<https://t.me/JSforEntDev>
- Github accounts to send PRs with complete tasks:
[@kos33rd](#), [@AVVlasov](#)
- News, announcements, resources and useful links:
<http://moodle.innopolis.university>