**Section 21**: Babel

Why do we need Babel, how to use it, etc ?

Babel is Javascript Compiler.

Compiler is a program that takes source code written in a programming language with higher level of abstraction and transforms it to the lower level language.

C++ 🡪 Assembly

Java 🡪 Java Bytecode

ES6 🡪 ES5 (We can compile ES6 to ES5)

Programming languages can be either compiled or interpreted.

**Lecture 284**: Compiled vs Interpreted Languages

Language cannot itself be either compiled or interpreted.

It depends on the implementation of the programming language.

Same programming language can be either compiled or interpreted depending on how it is implemented.

In compiled language implementation, there is compiler. This compiler takes in program code and converts it to machine code. And then we can run this machine code.

Each time when we run the program, we run the compiled version of the program.

If we need to do implement some changes in the source code, we need to run the source code through compiler again and create new compiled version.

If we create compiled version of the program for certain OS, for example, Windows, we will not be able to use this compiled versions on other platforms such as MacOS or Linux.

Usually, we will need to create different compiled versions for different operating systems.

In windows, such compiled versions have .exe versions.

In MacOS, it is .dmg extension and so on.

In Interpreted languages, there is an interpreter or engine. Interpreter takes in source program code and creates machine code.

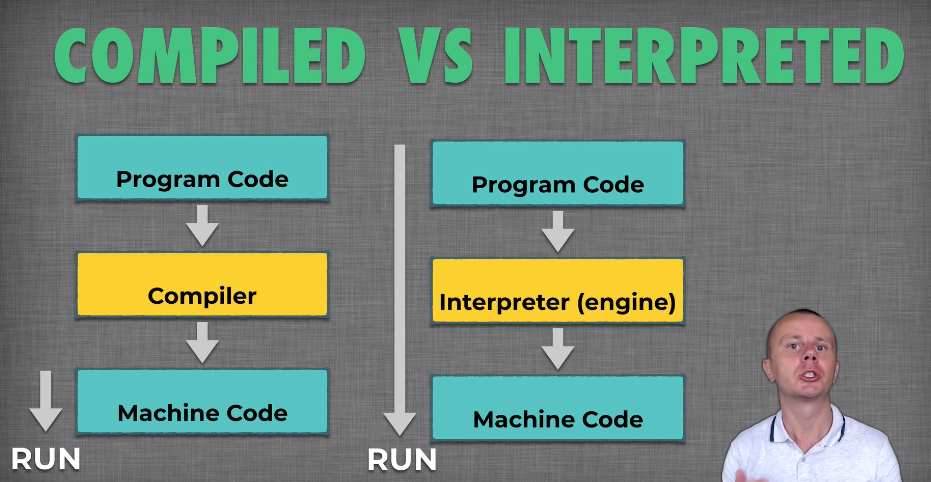
All of this happens during program execution (Runtime).

We have program code and interpeter which is always running. If we make some changes in program code, it is immediately executed by interpreter.

We need to keep our interpreter running while using our program and this will consume additional resources.

Interpreter is compiled version of another program written in one of the compiled programming languages.

There are many different programming languages that are usually implemented as interpreted languages, JS, python, php.



Sometimes program code is transformed from 1 programming language to another with the same level of abstraction.

It is called as transpiling.

**Lecture 285**: Transpiler

It is compiler that translates code from one programming language to another with the same level of abstraction.

We can transpile Sass to CSS  
CoffeeScript 🡪 Javascript

Python 🡪 Javascript

PHP 🡪 Javascript

We convert syntax of one programming language into the syntax of another progrmaming language.

Features of both of the programming languages are the same.

There are many different programming languages and all of them different levels of abstraction.

* JavaScript, Python, PHP
* C++, Java
* Assembly, Bytecode
* Machine Code

Final goal of a program is to be executed on the processor.

Processors understand only machine language or machine code.

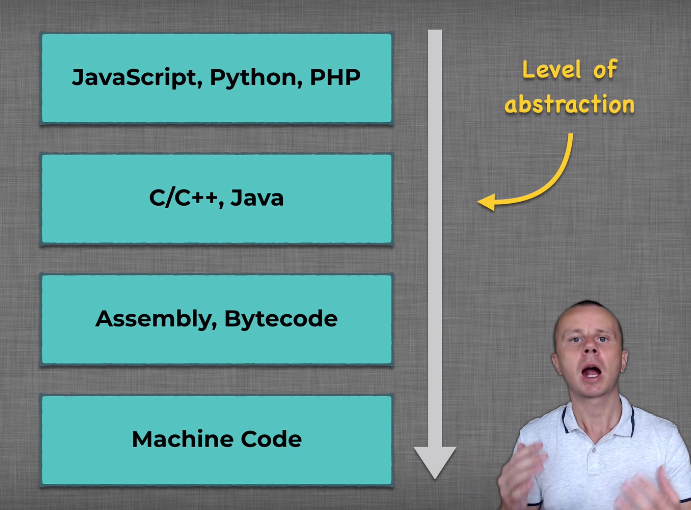
Each program must be finally compiled into machine code and executed using processor.

Some languages are located closer to machine code while some are far away.

Assembly languages and bytecode are most close to the machine code.

C, C++, Java have higher level of abstraction than assembly or byte code and they need to be compiled to assembly or bytecode.

Languages such as JS, PHP, Python are located at the top of the stack. This means they have highest level of abstraction as compared to other languages, and are located farthest from the machine code.



Program in these languages must go through various stages to finally run it in the processor.

If we transform code between languages at the same level of abstraction, then we call it transpiling.

If we go down the stack, then it is compiling.

Example of compiling is transforming Javascript into machine code.

**Lecture 287**: ES5, ES6 and Javascript

ES5 is short hand for 5th version of ECMAScript language specification standard release in 2009.

ecma-international organizations

ES6: created in 2015.

JavaScript is most popular implementation of the ECMAScript standard.

Developers of javascript language take ecmascript standard and implement all features described in the standard.

There are other implementations of the Ecmapscript, for example V8, SpiderMonkey, ActionScript.

V8 is the engine that is used in Node.js and Google Chrome. It takes Javascript code and converts it into machine code.

SpiderMonkey is used in Mozilla firefox and MongoDB.

It converts Javascript code into Bytecode.

ActionScript is used in the Adobe Flash Player.

There other implementations of the EcmaScript standard as well.

EcmaScript is the standard and this standard defines different features, syntax of the language.

Javascript is one of the implementation of EcmaScript standard.

Other implementations are V8, SpiderMonkey and ActionScript.

**Lecture 288**: What Babel does?

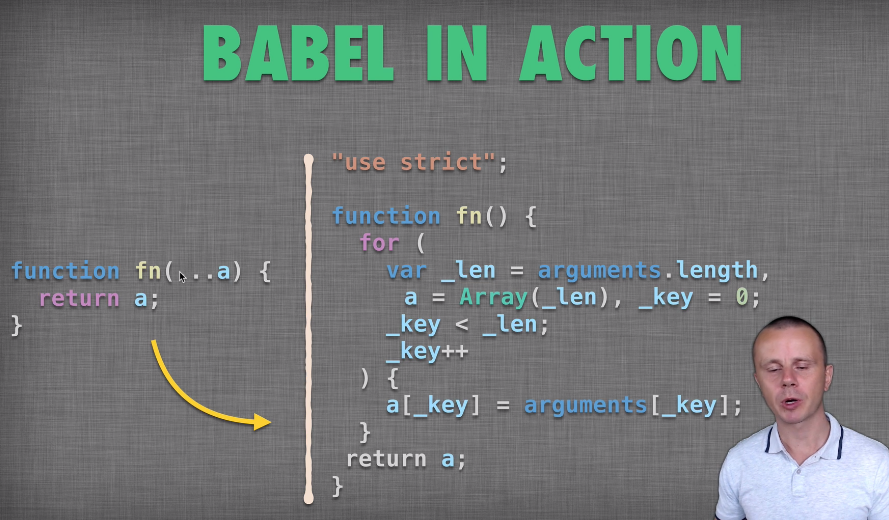
Babel transforms Javascript to Javascript.

Most common usage is to transform ES2015 (ES6) and above to ES5.

Babel is both compiler and transpiler.

Babel combines the features of compiler and transpiler.

REST operator is a new feature in ecmascript and it was absent before ES6.



If Babel takes next generation Javascript feature and converts it back to the previous Javascript version, like in the above example, then we can call it Compiler.

But if it takes Javascript feature that has the same syntax, for example, Javascript classes which are syntactic sugar for function constructor, then we can call it transpiling.

**Lecture 289**: Why do we need Babel ?

Usually browser requests file from server using HTTP protocol. Suppose in this file there are script links to the javascript scripts that are located on the server.

The browser makes request to the server and server has this JS file and it sends it back to the client.

What if this javascript file contains features from ES6 and later ?

Like “let”, “const”, arrow functions, rest, spread operators and so on.

What if the browser does not support those features ?

Suppose the browser supports ES5 features and does not support ES6, ES7, … features.

In this case, when browser will try to execute, download JS file, we will simply get an error,

Uncaught SyntaxError: Unexpected identifier

There are many browsers and not all support advanced JS features like ES6 and above.

In this case, on server side we need to perform some actions before passing JS file to the client.

If the browser does not support ES2015 features, then server needs to take Javascript file and convert it into ES5 syntax.

This should be done before passing JS document to the client browser. And only after that, when browser requests Javascript file, server will return converted Javascript file.

In this case, when browser will execute the javascript file on its side, we will not get any errors.

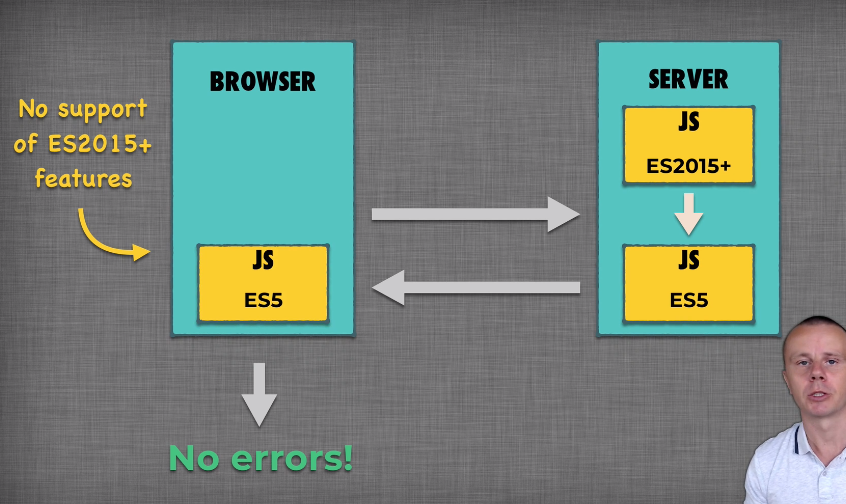
‘let’, arrow function were introduced in ES6.

Both are supported by Google Chrome.

With Google Chrome, we do not need to do any tranformation of the javascript file on the server side.

Internet Explorer does not recognize ‘let’ and arrow function syntax.

Try with microsoft edge?



For internet explorer, we need to perform such kinds of conversion.

Each browser has its own javascript interpreter.

**Lecture 290**:

There are many different implementations of the Ecmascript standard.

Some of them are Javascript language, spidermonkey, v8 engine, etc.

When we write Javascript code, we need to ensure that consumer of this code, for example, client, browser, node.js server will understand syntax of this code.

Import data from Google Analytics account. We can import data about users of our application, analyse coverage of features of browsers of our users,

**Lecture 291**:

When we compile ES6+ down to ES5, we support minority of users.

Code size will increase.

3 lines of code in ES6 could convert to more than 10 lines of code in ES5.

**Lecture 292**: ES6 Features Performance

To open a directory in vscode, we can use “code .”

babeljs.io: to convert ES6 to ES5 code.

‘arguments’ field is an object. We have to convert it to an Array.

function fn(…a){ // ES6

return a;

}

Let’s compare the performance of the above ES6 code and its ES5 version.

console.time(“Function 1 with ES6 features”)

fn(1, 2, 3)

console.timeEnd(“Function 1 with ES6 features”)

// same label

This runs in roughly 0.137ms

Console prints this :

Function 1 with ES6 features: 0.137ms

Lets call the function 1 million times in a for loop.

Function 1 with ES6 features: 18.724ms

ES5 version of the function: 1 million times

Function 1 with ES5 features: 28ms

**NOTE**: Babel always uses “use strict” option.

**Lecture 294**:

Built in Google Chrome performance feature

Performance tab, click on Record.

Refresh the page and then stop profiling.

Then go to Sources tab.

If we click on the file on the left, we can see how much time was required to execute both the statements.

This feature does not show us how much time was needed to execute each statement inside of the loop.

We will need to sum the values to find the time required to execute entire for-loop.

Performance Profiling feature in Chrome.

Doubt here.

**Lecture 295**:

Compare performance of destructuring

* Node.js
* Chrome

Compare ES6 and ES5 with both.

let [a, b, c] = array

ES5 version of destructuring performs better than ES6.

Not all features introduced in ES6 are fast.

This may change in the future when javascript interpreters will update their internal algorithms.

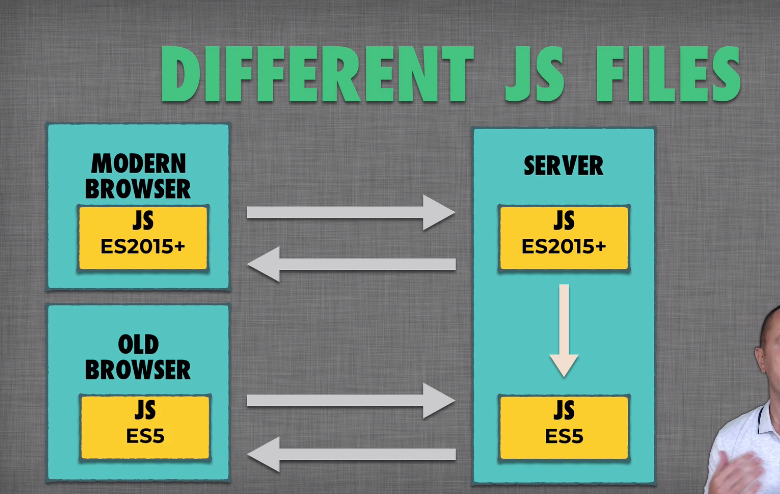
ES6 syntax is shorter but not always faster.

**Lecture 297**: What is User Agent?

Load compiled version only for old browsers.

Difference in ms.

File size increases when we compile ES6 down to ES5. Only send the compiled code to browsers which need this code.



We can detect browser(client) on the server using User Agent string.

Every time a browser sends an http request to the server, it sends a user agent string.

User agent is the information that is sent from the browser to the server and it contains information about browser version and Operating System of the computer where the browser is installed.

There are different ways to check this user agent of our browser.

Search “check user agent” on Chrome.

Using user agent string, we can detect user browser and OS.

window.navigator.userAgent

Each browser has global object “window”

How can we use this user agent in order to determine which Javascript file to send to the client ?

**Lecture 298**: Serve Different JS files. 3 options

* Decide on the server which file version to send to the browser
* Send same JS file to all browsers and use browser/feature detection
* Manipulate DOM on the client and inject link to certain JS file using browser/feature detection.

**Lecture 299**:

Option 1:

Decide on the server which file version to send to the browser

<body>

<script src= “abc.js”> </script>

</body>

Browser sends HTTP request and user agent is included in this request.

On the browser side, there is HTTP proxy server that intercepts this http request.

On this proxy server, we check the browser version. Depending on the browser version, we serve different versions of the same file.

Browser just requests one file. It does not know about the difference between different versions of the file.

Drawbacks: very easy to change user agent on client side.

There are plenty of different browsers in the world.

On the server, we will need to keep updated list of user agents and check each user agent based on this list. We can use external service for this.

**Lecture 300**: Option 2

Here we will send same JS file to all browsers (client) and use browser/feature detection.

This file will contain both ES6 and ES6 🡪 ES5 code.

Everything is in one file.

We can either check the browser of certain feature on the client. Browser check is not reliable.

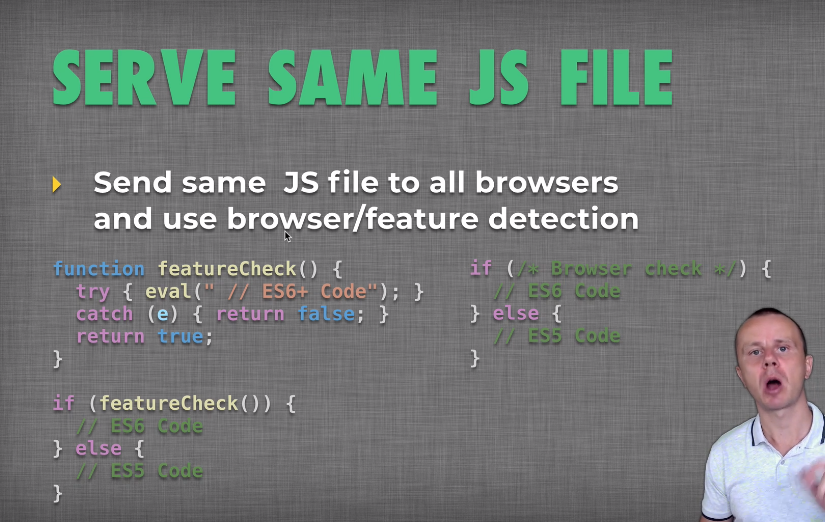
We can do it using if-else statement.

We can use navigator.userAgent property to perform certain checks.

If the checks are passed, then we execute ES6 piece of code, else we execute ES5 piece of code.

This is not reliable.

Better approach is to use feature detection.



We can check presence of certain ES6 feature on the browser. We can do this using try-catch approach.

In try, we can evaluate small piece of code in ES6.

If the browser does not produce any error, we can go on and use this feature.

Else we need to use compiled version.

var userAgent = navigator.userAgent;

if(userAgent.indexOf("Chrome") > -1 ){ // check presence of word Chrome.

console.log(`Hello from ES6`) // `` is ES6 feature.

} else {

console.log("Hello from ES5")

}

Run this in both Chrome and IE.

The above is unreliable as the user agent could change because of change in OS or browser.

Will need to maintain a table as discussed before.

**Lecture 302**: Detect feature

We will need to check whether or not template literals are supported on the browser and we will need to use try-catch approach for this.

function featureCheck(){

try{

eval("`Test String`")

}

catch(e){ // in case browser produces syntax error

return false

}

return true

}

if(featureCheck()){

console.log(`Hello from ES6`)

} else {

console.log("Hello from ES5")

}

The above gives “Invalid Character” error in Internet explorer.

In this, console.log(“Hello from ES6”) we use “” instead of `` and it works fine in internet explorer.

Feature detection is supported in internet explorer.

With internet explorer, if we use ES6 syntax in the JS file, then it gives an error.

We cannot use ES6 code in raw statements but can pass it in eval() method.

**Lecture 304**:

Conditionally load different javascript files from the client.

Manipulate DOM on the client and inject link to certain JS file using browser/feature detection.

We can achieve this using DOM manipulation.

Initially we load just a small script to the browser. This script checks browser version or feature support. If browser matches certain conditions, then we insert into the DOM link to one of the javascript files located on the server.

If the browser does not matches conditions, then we insert link to another js file on the server.

New function that will add additional <script> tag into the body of the html document.

Depending on the feature check, we can insert different links.

Lets create a file check.js and 2 other files, es6.js and es5.js

The function insertScript() will be executed by all the browsers and all of them must execute this function. This means that it must be written using ES5 syntax.

Create the es5.js and es6.js files in the src directory.

In the html file,

<body>

<script src= “check.js”></script>

</body>

function featureCheck(){

try{

eval("`Test String`")

}

catch(e){ // in case browser produces syntax error

return false

}

return true

}

function insertScript(type){

var el = document.createElement("script")

el.src = "src/" + type + ".js"

document.body.appendChild(el)

}

if(featureCheck()){

insertScript("es6")

} else {

insertScript("es5")

}

<script> tag means execute javascript code. The output comes one the console.

**Lecture 305**: How Babel works ?

3 stages:

* Parsing
* Transforming
* Generation

Babel takes js file as input and this could have ES6+ features.

Then it parses this file and splits it into small pieces of code.

Then we transform those pieces of code and finally we generate output file.

1 file as input as 1 file as output.

Default Babel Behavior:

function babel (code) {

return code

}

// just return the code without any modification.

Babel uses “uses strict”

It requires declaration of all variables that are used in the code.

**Lecture 307**: Install Babel

We can either install babel locally in the project directory or we can install it globally.

Locally:

npm install --save-dev babel-cli

Globally:

npm install -g babel-cli

babel-cli is the main babel package that has lots of dependencies.

--save-dev saves this package as development dependency.

We will talk about development and production dependencies later.

We can also install this package using yarn as well.

-g means that the package will be installed globally on the computer.

After this installation, we will have access to babel command in the terminal.

Local babel installation in each project is recommended approach.

To install babel-cli, we need to have node.js and npm.

babel-cli has a lot of dependencies.

babel src/index.js -o --out-file src/babel.js

The output file will be same as input file. This is the default beahvior of Babel.

**Lecture 309**: Plugins and Presets

Plugin is an independent programming module that is dynamically connected to the main program.

In Babel, plugins are connected at the transformation phase.

There are various babel plugins available.

It is not convenient to add plugins independently to our program.

Preset: Set of plugins

babel-preset-es2015: this present contains many different plugins related to es2015 syntax.

For example, es2015 classes, spread operator and so on.

Presets can be combined in other presets called as environment presets.

babel-preset-env:

This preset contains all ecmascript features starting from es2015.

This preset is updated each time new version of ecmascript is released.

babel-preset-env also contains packages such as browserslist, semver and invariant packages.

These packages help to target specific browsers using babel.

There are also other presets called as stage presets.

Stage 0: Initial idea

Stage 1: Proposal

Stage 2: Draft-initial specification

Stage 3: Candidate - complete specification

Stage 4: Finished

ECMAScript Technical Committee proposal categories

babel-preset-stage-0

babel-preset-stage-1

babel-preset-stage-2

babel-preset-stage-3

There are different ecmascript features that are not yet included in the standard.

Each feature before becoming standard goes through different stages.

Stage 0 is the initial idea of the feature.

Stage 2 is draft and the feature has initial specification.

Stage 3: feature becomes a candidate to be included in the standard.

Stage 4 means that the feature is included in the ecmascript standard.

There is no preset for stage 4 because stage 4 is already standard and feature will be included in ecmascript, for example es2018 or es2019 preset.

If we want to use in our project some ecmascript features that are not yet included in the standard, then we must add those stage presets.

Each stage preset contains all next stage presets

* Its enough to install only Stage 0 preset.

If we add stage 0 preset, then this preset automatically adds all features available in preset 1, 2, and preset 3.

Earlier stage contains all later stages.

If we add just stage 3 features, then features from stage 0, 1 and 2 will not be added.

We can also add a specific feature using babel plugin.

**Presets installation**:

Locally:

npm install --save-dev babel-preset-env

(saved as development dependency)

Plugins and Presets must be installed locally in the project even if babel-cli is installed globally.

If we install them globally, they won’t work. Why ?