Intro – Babel

Topics:

- Packet Managers
- Introduction – NodeJS.
- Operators of ES6 – Spread & Rest

Intro – NodeJS:

nodejst.org → ***Download NodeJS (LTS version) and install node.***

Every time we have seen the output of the javascript code, we need a html file in which our javascript file is linked so that we can see the out. But after adding node, we can see the output directly in the terminal. This is happening due to NodeJS. This gives us flexibility and freedom in JavaScript world.

In our browser, we were able to call the document.getElementById, here in the node JS code we would not be able to do that, because whatever JavaScript code which is supposed to be running through node.js, we won't be using any browser oriented specific script.

Ex: 01

```
let a = 12
let b = 23
console.log(12 + 23)
```

After writing this code, we need to go the specific directory using the terminal by specifying the path and then enter the file name with extension see the output.

Directory:\projects\attainu\script.js

If we had written a word 'document' object, it would not work because NodeJS doesn’t have document object.

Similar to the python, if we write node any file name in the terminal, node will run the file.

Package Managers:

There are some third-party libraries like 'pip' or 'NumPy' which was install and use it in our code to simplify our development process. Node internally comes along with a package manager know as NPM (node-packet-manager). In npmjs.com, all the third-party open-source libraries are hoisted. We can fire some command and use it in our code.

Let’s search for 'Nano ID' – a unique string ID generator for JavaScript. To use this we will install it using **`''npm install nanoid''`**.

We can see, the terminal has searched the internet and installed the nanoid. It created a folder known modules in which we have see the Nano ID folder in which all it’s files are mentioned. This is crypto graphic module of NodeJS itself. Now, in our script.js file, we need to **`import {nanoid} from 'nanoid'`**.

We need to call the function to generate a string.

```
import {nanoid} from 'nanoid'.

let uniqueId = nanoid()
console.log(uniqueId)
```

As we have used import function out a module it would show an error. So we would change the code to,

```
const {nanoid} = require('nanoid')

let uniqueId = nanoid()
console.log(uniqueId)
```

This is one example of how to import different libraries into our code and use it accordingly.

There are certain times in our project when we need to do the code to do something. You can either create your own API for a particular project you are working on or you can import the package directly into your project. Everyone started doing this, so to manager all the packages, **package manager** came in. We have to install NodeJS into our system to run the javascript code without the html file.

Babel:

Babel was a transpiler and over the time got converted into a compiler. It converts your code and adds quite a lot stuff for you. You can write using any new feature of JavaScript (ES6, ES7 and so on), and Babel will convert it to ES5, allowing you to target the larger audience.

One of the versions is using the chrome v89 and another person using v45, both will be able to use it. The frame will take place automatically as babel would already be installed.

let abc = 123

const xyz = 100

The const and let should be changing to es5 version. We have to install babel into our project. Using

```
npm install @babel/core @babel/cli @babel/present-env
```

on install babel multiple files get created in our directory. One last thing would be have “babel.config.js” as it is specific to Javascript. After doing this, we have to put in {} and inside this write **`"presets": ["@babel/env"]`**.

To access @babel/cli, we have to type

```
.\node_module\.bin\label script.js –out-file main.js
```

It will create another main.js file. In the file we can see that let and const were converted to var and the class Animal was converted to function. We have basically converted ES6 code to ES5 equivalent; everything with just a matter of one command.

Rest:

An add() function should be able to take n number of arguments.

```
function add(...args) {
    console.log(args)
}

add (10)
add (10, 20)
```

We can see the output as list.

[10]

[10, 20]

Rest collects multiple elements and condenses them into a single element. It is not entirely an array, but it is inheriting from the array.

```
function add(...args) {
    console.log(args)
    let sum = 0
    args.forEach(item => sum += item)
    console.log(sum)
}

add (10)
add (10, 20)
```

Output would be 30.