ACADGILD

LEARN. DO. EARN

FRONTEND DEVELOPMENT

(WITH ANGULAR JS)

ACADGILD

ES6 I

Session - 2

LEARN. DO. EARN

Agenda

ACADGILD

Sl No.	Agenda Title
1	What is ES6?
2	Scooping
3	Destructing
4	Promises
5	Arrow Functions

What is ES6?

What is ES6?

- **ECMAScript 6** is the sixth major release of the ECMAScript language specification.

- **ECMAScript** is the “proper” name for the language commonly referred to as JavaScript.

- **Now, Why should we learn ES6?**
- Evolving JS has the leverage to add or change the semantics of the platform in the way that no other strategy credibly can.
- So we go awesome changes in ES6 which will help us.

Do all Browsers support ES6?

ACADGILD

- ES6 support is coming along well in major desktop browsers, with Chrome at more than 95 percent compatibility, Edge at 83 percent and Safari at 58 percent.
- But It doesn’t mean you cannot start with ES6, there are lot of pre-processor which will compile ES6 into ES5 which all browser supports.
- We can use Babel or many others compilers are there.
- So this is right time to start ES6.
- We will try to cover all basics and important features of ES6.

Some Basic Features of ES6

ACADGILD

- Scoping
- Destructuring
- Promises
- Arrow Functions
- Sets & Maps
- Rest parameters

- For more details about features and browser compatibility try compat-table. There are lot of things you can go in deep.

Some Basic Features of ES6

ACADGILD

- Some other basics, we should know before going in deep.

- Default parameters in ES6:
  - Remember, how we are used to define parameters in ES5:

```
var comp = function(name, regid) {
    var name = name || 'Acadgild'
    var regid = regid || 90125
}
```

- But in JS, 0 is falsy value, it would default to the hard-coded value instead of becoming the value itself, so we just ignored this flaw and used the logic OR.

Some Basic Features of ES6 (Contd.)

- But In ES6, we can put the default values right in the signature of the functions

```
var comp = function(name='Acadgild', regid=90125) {
    Do something...
}
```

- Template Literals in ES6:
  - As we are using template literals till now in ES5:

```
var name = 'My name is ' + first + ' ' + last + '.'
```

- In ES6:

```
var name = `Your name is ${first} ${last}.`
```

Some Basic Features of ES6 (Contd.)

ACADGILD

- **Multi-line Strings in ES6:**
  - See ES5 syntax:

```
var intro = 'I am Avnesh Shakya,\n\t'
    + 'And going to share ES6 with you.\n\t'
    + 'It will be more fun.\n\t'
```

- Now see in ES6:

```
var intro = 'I am Avnesh Shakya,\n\
    'And going to share ES6 with you.\n\
    'It will be more fun.'
```

Scoping

ACADGILD

1. Scoping:

- In ES5, variables are either globally or locally function scoped. But in ES6, we can use block scoping.

- There are two types of ***variables*** in ES6:

i) ***let***

ii) ***Const***

i) **Let**:

It's sounds weird when we see let instead of **var**. But it's allows to scope the variable to block. To understand ***let,*** first we need to keep in mind that ***var*** creates function-scoped variable.

```
function testScope() {
  var name; // variable declaration is hoisted to the top
  if(true) {
    name = "Avnesh";
  }
  console.log(name); // Avnesh
}
```

Scoping (Contd.)

ACADGILD

But now using **let**,

```
function testScope() {
  if(true) {
    let name = "Avnesh";
  }
  console.log(name); // ReferenceError: name is not defined
}
```

As ***let*** is inside a block, name is only seen inside that block.

Scoping (Contd.)

ACADGILD

ii) **const**:

- Till now we were missing const value in JavaScript.

- With ES6 we will be able to create constants and make sure its value won't be changed during the application execution.

  const API_URL = http://acadgild.com/user

- const works like let, but the variable you declare must be immediately initialized, with a value that can’t be changed afterwards.

Destructuring

ACADGILD

- Destructuring is the process of assigning the property values of an object to a local variable.
- Initially it may hard to understand but it’s awesome feature in ES6.
- In other words, the destructuring assignment syntax is a JavaScript expression that makes it possible to extract data from arrays or objects into distinct variables.

```
/**
  * extract data from arrays or objects
*/
var foo = [1, 2, 3];
var [one, two, three] = foo;
// one => 1, two => 2, three => 3
var {a, b} = {a:1, b:2};
// a => 1, b => 2
```

Destructuring (Contd.)

ACADGILD

- In JavaScript, we can do this:

```
var acadgildGlobalConfig = {
    apiUrl: 'www.acadgild.com/api',
    data: 'some value',
    methodType: 'POST'
};

// now use in functions
function makeAjaxRequest(config){
    var url = config.url;
    var method = config.methodType;
    var data = config.data;
    $.ajax(url, method, data );
}
```

Destructuring (Contd.)

ACADGILD

- But is ES6 using destructuring,

```
// and in one of our function
function makeAjaxRequest(config){
    var { url, methodType, data2 } = config;
    $.ajax(url, methodType, data2 );
}
```

- It might take some time to get use to the destructuring assignment syntax, but this is awesome feature in ES6.

Promises

ACADGILD

- Promises provide a mechanism to handle the results and errors from asynchronous operations.
- You can accomplish the same thing with callbacks, but promises provide improved readability via method chaining and succinct error handling.
- Promises are currently used in many JavaScript libraries.

```
/**
  * Promises are used for deferred and asynchronous computations.
  * A Promise represents an operation that hasn't completed yet, but is expected in
the future.
  */
var foo = new Promise(function (resolve, reject) {
  //Check if the current timestamp is an even number and resolve
  if (Date.now() % 2 === 0) {
    //Pass a status code of 200 to the success callback function
    resolve(200);
  } else {
    //Pass a status code of 404 to the failure callback function
    reject(404);
  }
});
```

Promises (Contd.)

ACADGILD

```
/When the promise has successfully resolved, execute the following
//callback function
foo.then(function (status) {
  console.log("Successfully resolved: " + status);
});

//When the promise is rejected i.e. an error, execute the following
//callback function
foo.catch(function (status) {
  console.log("An error occurred: " + status);
});
```

Some methods of Promises:

1. Promise.resolve(value)
2. Promise.cast(value)
3. Promise.race(value)
4. Promise.all(value)

- Promises are a first class representation of a value that may be made available in the future.

Promises

ACADGILD

- **A promise can be:**
  - fulfilled —promise succeeded
  - rejected —promise failed
  - pending—not fulfilled or not rejected yet
  - settled—fulfilled or rejected

- Every returned "promise object" also has a "then" method to execute code when a promise is settled.

```
new Promise((resolve, reject) => {
  // when success, resolve
  let value = 'success';
  resolve(value);
  // when an error occurred, reject
  reject(new Error('Something happened!'));
});
```

Arrow Functions

ACADGILD

- Actually Arrow functions is not going to fundamentally change anything.
- **Arrow functions provide two features:** lexical scoping of the this keyword and less ceremony when defining an anonymous function.
- Without arrow functions, every function defines a this value. No more will you need to reassign this.
- Using arrows functions in ES6 allows us to stop using that = this or self = this or _this = this or .bind(this). For example, this code in ES5 is ugly:

```
var _this = this
$('.btn').click(function(event){
  _this.sendData()
})
```

And in ES6, it's became so easy using arrow:

```
$('.btn').click((event) =>{
  this.sendData()
})
```

Arrow Functions (Contd.)

ACADGILD

- You might be used coffeescript, then it's easy to understand and even it helps to save time to right code.

```
let foo = ["Hello", "World"];

//single arguments do not require parenthesis or curly braces.
//The return statement is implicit
let bar = foo.map(x => x.length);

// ES5
var bar = foo.map(function(x) { return x.length; });

//multiline functions require curly braces
//no arguments expect parenthesis
let foobar = () => {
    console.log("Hello");
    console.log("World");
};
```

Arrow Functions (Contd.)

ACADGILD

```
// ES5
var foobar = function() {
    console.log("Hello");
    console.log("World");
};

//Returning object literal. Requires Brackets.
let quux = () => ({ "myProp" : 123 });

//ES5
var quux = function() {
    return { "myProp" : 123 };
};
```

- Try some other examples for more understanding.

Drawback of Arrow Function (Contd.)

ACADGILD

- Some limitations of using Arrow function:
  - It's cannot be used as a constructor and will throw an error when used with new.
  - This means that you cannot change the value of this inside of the function.
  - It remains the same value throughout the entire lifecycle of the function.
  - Regular functions can be named.
  - Functions declarations are hoisted (can be used before they are declared).


Questions?

ACADGILD

THANK YOU

For more details contact us at:

Support - +91 91 8884666874

Email us at - support@acadgild.com

LEARN. DO. EARN