Can JSX be directly read by the browser? What are some of the differences between ES5 and ES6 syntax?

No. Because browsers only read regular JS objects. For web browsers to read a JSX file, the JSX file must be transformed to JavaScript Object, tools like Babel can transform jsx to js.

ES5	ES6
Only **var** keyword is used to define a variable.	**let** and **const** are newly added keywords can also be used to define a variable.
Arrow functions can not be used.	Arrow functions can be used

Object Destructuring must be done manually, e.g.:
`var spec = { speed: 2.6, cores: 12 }`
`var speed = spec.speed`
`console.log(speed) //2.6`
	Object Destructuring is easier.
	`var {age,name} = {age:12,name:"John"}`
	`console.log(age)`

module.exports is used to export module.	**export** keyword is used to export module.

+ must be used for string interpolation, e.g.:
`var age = 12`
`console.log("Adam is " + age + " years old")`
	Backticks `` , Template Literal, can be used for string interpolation, e.g.:
	`var age = 12`
	`console.log(`Adam is ${age} years old`)`

ES5	ES6

Spread operator(...) is not available	Spread operator(...) is available
	`var spec1= {speed:"3.2Ghz",cores:24}`
	`var pc={...spec1}`
	`console.log(pc)//{speed: '3.2Ghz', cores: 24}`

Import keyword is not available to import a module `var Register=require("./Register")`	Import keyword is available to import a module `import Register from './register';`

Shorthand for defining objects with keys and the variable names are not available	Shorthand for defining objects with keys and the variable names are available
	`var cores=24`
	`var speed=3.4`
	`var spec={cores,speed}`
	`console.log(spec.cores)//24`

Does not support Promises	Supports Promises
There is no class keyword in ES5	ES6 supports classes

ES5	ES6

ES5 uses **module.exports** to export objects	We can use **export default** to export a module, but remember a module can't have multiple **default** exports
	
	`export default Register;`
	
	To import it now from another component we do:
	
	`import Register from './Register';`
	
	We can export multiple objects:
	
	`function Register() { return <div> <h1>Register</h1> </div> }`
	
	`export var user1 = { name: "John", age: "82" }`
	`export var user2 = { name: "Adam", age: "46" }`
	
	`export default Register;`

ES5	ES6

	Use **{}** to import multiple exported objects those are exported **without** using the **default** keyword:
	`import Register,{user1,user2} from './Register';`
	But you don't have to use **{}** to import a single object that is exported without using the default keyword:
	`import user1 from './Register';`
	You can change the function's name that is exported using the **default** keyword when importing:
	`function Register() { return <div> <h1>Register</h1> </div> } export default Register;`
	And you can now import as any name:
	`import RegisterUser from './Register';`

ES5	ES6

	But you can not change the name of the exported object that is exported without using the **default** keyword:
	`export var user1 = { name: "John", age: "82" }`
	
	You can not import it like this:
	`import user12 from './Register';`
	
	You must use the same exact exported name when importing:
	`import user1 from './Register';`

	So exports without a **default** tag are Named exports and exports with the **default** tag are default exports. Named exports allow multiple exports in a single file, e.g.:
	`export function MyFunction() { return <h1>Ok</h1> }`
	`export function MyFunction2() { return <h1>Ok</h1> }`