Chapter 1

- Chapter 1
  - Babel
    - Installation
    - Transpile
  - JSX Basics
    - Important JSX Rules
  - ES6 Basics
    - var, const and let
    - Arrow Functions
  - JSX Data Binding
  - JSX Form Submit
  - JSX Arrays

Babel

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. Here are the main things Babel can do for you:

Installation

```
yarn global add babel-cli
yarn add babel-preset-react babel-preset-env
```

Transpile

```
babel src/app.js --out-file=public/scripts/app.js --presets=env,react
```

above command will convert the React JSX code in `src/app.js` into ES5 compatible app.js at `public/scripts/app.js`

```
babel src/app.js --out-file=public/scripts/app.js --presets=env,react  --watch
```

to watch for any change in `src/app.js` and automatically transile the file

JSX Basics

Important JSX Rules

1. We can only return one top-level element from a given component. This is usually known as a parent element and is used to group the content.

```
var template = <h1>My React app</h1><p>Test</p>; // not allowed
```

In above template adjacent elements are placed without parent container, so JSX will give error

Above can be avoided by using below

```
var template = <div><h1>My React app</h1><p>Test</p></div>;
```

above can also be written as

```
var template = (
  <div>
    <h1>My React app</h1>
    <p>Test</p>
  </div>
);
```

2. Some elements in HTML do not have a closing tag. In React JSX, every tag, including those with no closing tags, must be closed. If you have an element that doesn't have a closing tag, you have to add a slash at the end (e.g., `<hr/>`). Noting special here, simple HTML rule

3. A React component must be capitalized. Component names that do not begin with a capital letter are treated like built-in components, and it results in strings ("div", "span"...). When the component name is capitalized, it is treated as an identifier instead of a string.

4. To include JavaScript expressions in JSX, we need to wrap them in curly braces. Content between the opening and closing curly braces will be evaluated as JavaScript.

5. The term "class" is a reserved keyword in JavaScript. In React, we must substitute className for class.

    1. JSX DOM Elements attributes
    2. JSX Supported Events

6. undefined, null and false are ignored by JSX

undefined is returned implicitly by function, if expression evaluates to undefined nothing will be displayed

```
function getUserLocation(location) {
  if(location) {
    return <p>Location : {location}</p>;
  }
}
```

```
var template2 = (
  <div>
    <h1>{user.name}</h1>
    <p>Age : {user.age}</p>
    {getUserLocation(user.location)} // will not print anything if the expression
evalues to Undefined
  </div>
);
```

Explaination of above function

```
function getUserLocation(location) {
  if(location) {
    return <p>Location : {location}</p>;
  }
}
```

is equivalent to

```
function getUserLocation(location) {
  if(location) {
    return <p>Location : {location}</p>;
  }
  return undefined;
}
```

7. Javascript objects are not valid as a React child

```
var user = {
  "name" : "Manish",
  "age" : 31,
  "location" : "Mumbai"
}

var template2 = (
  <div>
    <h1>{user}</h1>
    <p>Age : 26</p>
    <p>Location : Mumbai</p>
  </div>
);
```

we can't use `user` object directly in react template

below example is a valid template

```
var template2 = (
  <div>
    <h1>{user.name}</h1>
    <p>Age : {user.age}</p>
    <p>Location : {user.location}</p>
  </div>
);
```

ES6 Basics

var, const and let

- Declare and Assign

Type	re-declare	re-assign
var	Y	N
let	N	Y
const	N	N

```
var nameVar="Manish";
nameVar = "XYZ";
var nameVar = "Piyush";
console.log('nameVar : ', nameVar); // will print Piyush

let nameLet = "Romeo"
nameLet = "Juliet"
// let nameLet = "xyz" // Not Allowed
console.log("nameLet :- ", nameLet)

const nameConst = "Frank";
// nameConst = "Gunther" // not allowed
// const nameConst = "XYZ" // not allowedvar a

console.log("nameConst :- ", nameConst)
```

- Scope
    - all are scoped to a function, i.e. not visible outside the function where it is scoped

```
function getPetName() {
  var petName = "JoJo";
  return petName;
}

console.log(petName); // not visible
```

- `let` and `const` and block level scoped, `var` defined inside the block is accessible outside the block it is defined within the same method

```
var fullName = "Manish Singh";
if(fullName) {
  var firstName = fullName.split(" ") [0];
  console.log("FirstName in Block :- ", firstName)
}
console.log("FirstName outside Block :- ", firstName);
```

In above example, although firstName is defined inside `if` block but it is still visible outside the block

```
const fullName = "Manish Singh";
if(fullName) {
  let firstName = fullName.split(" ") [0];
  console.log("FirstName in Block :- ", firstName)
}
console.log("FirstName outside Block :- ", firstName); // will give
undefined
```

for `const` and `let`, the above example will give undefined as `firstName` is scoped within `if` block only

Arrow Functions

1. Arrow Function Expression (Shorthand syntax for one line operations) ES5 Syntax

```
const square = function(x) {
  return x * x;
}
```

can be written as

```
const sqArrow = (x) => {
  return x * x;
}
```

can further be reduced to

```
const sqArrow = (x) => x*x; // arrow function expression syntax
```

2. `arguments` object not bound with arrow function

```
const add = function(a, b) {
console.log(arguments);
return a + b;
}

add(4, 5); will print [4, 5] and then sum of [4, 5]
add(4, 5, 6); will print [4, 5, 6] even though method is declared with two params
and then sum of [4, 5]
```

but the `arguments` object is not visible in arrow function

```
const add = (a + b) => {
  console.log(arguments);
  return a + b;
}


add(4, 5) // arguments is not defined, will give error
```

3. `this` keyword not bound to arrow function

```
const user = {
    name : "Manish",
    cities : ["Mumbai", "Bikaner", "Varanasi"],
    placesLived : function() {
        console.log(this.name);
        console.log(this.cities);
        const that = this;
        this.cities.forEach(function(city) {
            // console.log("this.name :- "+this.name + " has lived in ", city); //
this is not accessible
            console.log("that.name :- "+that.name + " has lived in ", city);
        })
    },

    placesLivedArrow : function () {
        this.cities.forEach((city) => { // this refers to parent scope of function
i.e. the user object
            console.log(this.name + " has lived in ", city);
        })
    },

    placesLivedArrow2 : () => { // this arrow function doesn't bind its own "this"
value and goes up to the parent scope which is global scope (parent of user
object) and is undefined
        this.cities.forEach((city) => { // this here will be undefined
```

```
            console.log(this.name + " has lived in ", city);
        })
    },

    placesLivedArrow3() { // to avoid implementation array of placesLivedArrow2
        this.cities.forEach((city) => {
            console.log(this.name + " has lived in ", city);
        })
    }
}
```

1. `placeslived` ES5 method
   In `placeslived` ES5 method the `this` used in `forEach` is `undefined`
2. `placesLivedArrow` Arrow function for forEach
   keyword `this` used in forEach refers to parent scope i.e. `user` object, so the `this.name` will print the name
3. `placesLivedArrow2` Arrow function for property
   for `this.cities` the `this` keyword is not bound to `user` object but instead it is bound to `global` scope and hence will give `undefined` and cause error
4. `placesLivedArrow3` shorthand arrow function as property
   with new ES6 implementation, function can be declared without attribute and the resultant code will be as below

```
```
placesLivedArrow3: function() {
        this.cities.forEach((city) => {
            console.log(this.name + " has lived in ", city);
        })
    }
```
```

JSX Data Binding

- JSX doesnot has built-in data binding

```
let count = 0;
    const someId = "myId" // this can be used as expression in id attribute
in template
    const addOne = () => {
        count++;
        console.log("add count clicked :- "+count)
    };
    const minusOne = () => {
        count--;
        console.log("Minus one clicked")
    };
    const resetCount = () => {
```

```
        count = 0;
        console.log("Reset button clicked")
    };

    const template2 = (
    <div>
        <h1>Count : {count}</h1>
        {/*<button onClick={() => console.log("Arrow Clicked")}>+1</button>
*/}
        <button onClick={addOne}>+1</button>
        <button onClick={minusOne}>-1</button>
        <button onClick={resetCount}>Reset</button>
        </div>
    );

    ReactDOM.render(template2, appRoot)
```

for above example, every time a button is clicked, the count variable value changes but the variable in UI will not change as JSX doesnot have built-in data binding

- Manual Data Binding by re-rendering

```
let count = 0;
    const someId = "myId" // this can be used as expression in id attribute
in template
    const addOne = () => {
        count++;
        console.log("add count clicked :- "+count)
        renderCounterApp(); // on every click re-render the template
    };

    const minusOne = () => {
        count--;
        console.log("Minus one clicked")
        renderCounterApp();
    };

    const resetCount = () => {
        count = 0;
        console.log("Reset button clicked")
        renderCounterApp();
    };

const renderCounterApp = () => {
  const template2 = (
    <div>
      <h1>Count : {count}</h1>
      {/*<button onClick={() => console.log("Arrow Clicked")}>+1</button>
*/}
      <button onClick={addOne}>+1</button>
      <button onClick={minusOne}>-1</button>
```

```
        <button onClick={resetCount}>Reset</button>
      </div>
    );
    ReactDOM.render(template2, appRoot);
  }

  renderCounterApp();
```

Every time an event is called, value is updated and DOM is rendered, React will not re-render the whole component, but it will only update the `count` in DOM

JSX Form Submit

```
const app = {
  title: 'Indecision App',
  subtitle: 'Put your life in the hands of a computer',
  options: []
};

const onFormSubmit = (e) => {
  e.preventDefault(); // avoids full page reload on Form Submit

  const option = e.target.elements.option.value;

  if (option) {
    app.options.push(option);
    e.target.elements.option.value = '';
    render();
  }
};

const onRemoveAll = () => {
  app.options = [];
  render();
};

const appRoot = document.getElementById('app');

const render = () => {
  const template = (
    <div>
      <h1>{app.title}</h1>
      {app.subtitle && <p>{app.subtitle}</p>}
      <p>{app.options.length > 0 ? 'Here are your options' : 'No options'}</p>
      <p>{app.options.length}</p>
      <button onClick={onRemoveAll}>Remove All</button>
      <ol>
        <li>Item one</li>
        <li>Item two</li>
      </ol>
      <form onSubmit={onFormSubmit}>
```

```
        <input type="text" name="option" />
        <button>Add Option</button>
      </form>
    </div>
  );

  ReactDOM.render(template, appRoot);
};

render();
```

- `e.preventDefault();` written inside the `onFormSubmit` function will avoid the full page to be reloaded
- In above example, everytime `options` array is updated the DOM is re-redered via the `render` method to update the `options` list in the DOM

JSX Arrays

- `undefined`, `null` and `false` are ignored in the JSX array

  ```
  {
    [97,98,99, null, undefined, true, false]
  }
  ```

  is equivalent to

  ```
  {
    [97,98,99, true]
  }
  ```

  Render JSX In Arrays

  ```
  {
    [<p>a</p>, <p>b</p>, <p>c</p>]
  }
  ```

  above will give warning to add key to individual key/properties to optimize the rendering

  ```
  {
    [<p key="1">a</p>, <p key="2">b</p>, <p key="3">c</p>]
  }
  ```

- Use array to print template

for below array

```
const numbers = [55, 101, 2001]
```

below syntax can be used inside template

```
{
  numbers.map((num) => {
    return <p key={num}>Number: {num}</p>
  })
}
```

To create ordered list from above array

```
<ol>
  {/* map over app.options - This is comment inside JSX template */}
  {
    app.options.map((opt, idx) => <li key={idx}>{opt}</li>)
  }
</ol>
```