前端網站開發人員證書課程
(二) 進階網絡程式設計--專業React.js應用

1. Fundamentals of JavaScript and React

Presented by Krystal Institute

Lesson Outline

- Getting started with React
- Fundamentals of JavaScript Classes
- Introduction to ReactJS

What are the key benefits of using React JS

How to render elements in React

React Components and Types

React Props and State

Lifecycle of React Components

React Forms

Component interaction in React

Glimpse of Assignment

1.1 Get Started with React

1.1.1 React JS

Have you ever wished to create your own modern website or app with high performance and security? If you did, learning React will help you to do it.

- React JS is a **powerful front-end framework** based on JavaScript
- React follows a component-based design, which allows the developers to create a resizable web application or complex user interface by integrating small and isolated snippets of code
- This process may seem complicated at this point but by gaining expertise in the right prerequisites for React JS you can easily excel in it

1.1.2 Fundamentals of JavaScript

React makes extensive use of JavaScript as compared to other frontend solutions.

Let's brush up on some essential JavaScript concepts.

- How to create Classes?
- How to create objects?
- How to access the methods inside the class using the objects created?

1.1.2 Fundamentals of JavaScript

JavaScript Class

- A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object.
- Class Declaration syntax:

  class Classname {

  }

- Where *class* keyword is followed by a class name.
- A class definition includes the following:
  - **Constructors:** Responsible for allocating memory for the objects of the class.
  - **Methods:** Functions represent actions an object can take. They are also at times referred to as methods.

1.1.2 Fundamentals of JavaScript

Constructor

- A constructor() method is a special method for creating and initializing objects created within a class.
- The constructor() method is called automatically when class is initialized, and it was to have the exact name constructor.
- If the constructor() method is not added to the class, JavaScript will add an invisible and empty constructor method.
- A class cannot have more than one constructor.

```
var Polygon = class Polygon {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
}
```

1.1.2 Fundamentals of JavaScript

Class Methods

- Class Methods are the functions inside the class.
- Class methods are created with the same syntax as object methods.
- Use the keyword “class” to create a class.
- Add constructor() method.
- Then add any number of methods.

```
class Polygon {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
    test() {
        console.log("The height of the polygon: ", this.height)
        console.log("The width of the polygon: ",this. width)
    }
}
```

.

1.1.2 Fundamentals of JavaScript

Create Objects

- To create an instance of the class, use the new keyword followed by the class name. Following is the syntax for the same.
- Syntax:

  var objectname= new classname([arguments])
- Where,
  - The new keyword is responsible for instantiation.
  - The right-hand side of the expression invokes the constructor. The constructor should be passed values if it is parameterized.
- Example:

```
var obj = new Polygon(10,12)
```

.

1.1.2 Fundamentals of JavaScript

Access Class Methods

- A class's attributes and methods can be accessed through the object.
- Use the '.' dot notation (called the period) to access the data members of a class.
- Syntax:

  obj.functionname()

- Example:

```
obj.test();
```

where:

- obj → The objected, that has the instance of the class.
- test() → The method inside the class.

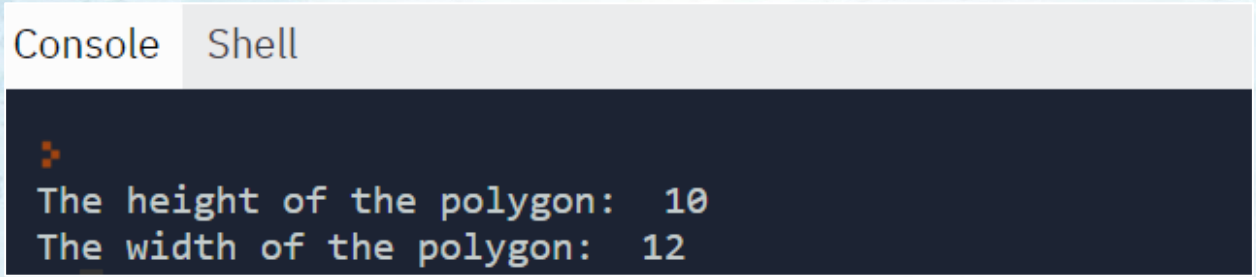
1.1.2 Fundamentals of JavaScript

Classes and Prototypes - Example 1

```
//class
class Polygon {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
   test() {
        console.log("The height of the polygon: ", this.height)
        console.log("The width of the polygon: ",this. width)
    }
}
//Object Creation
var obj = new Polygon(10,12)
//Accessing Function
obj.test();
```

1.1.2 Fundamentals of JavaScript

Classes and Prototypes - Example 1



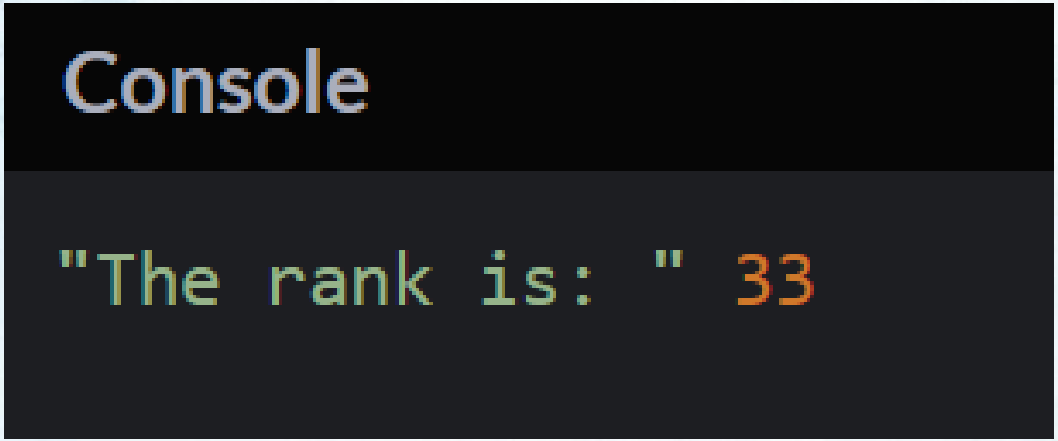
1.1.2 Fundamentals of JavaScript

Classes and Prototypes - Example 1

```
class student {
  // Constructor
  constructor(name, year, rank){
    this.n = name;
    this.y = year;
    this.r = rank;
  }
  // Function
  decreaserank(){
    this.r -= 1;
    console.log("The rank is: ",this. r)
  }
}
//objects creation
const student1 = new student("Tim", 2009, 34)
const student2 = new student("John", 2012, 45)
const student3 = new student("Tim", 2010, 76)
//Accessing functions
student1.decreaserank();
```

1.1.2 Fundamentals of JavaScript

Classes and Prototypes - Example 1



1.1.2 Fundamentals of JavaScript

Class Inheritance

There are two types of class:

- **Parent class / Super class:** The class extended to create new class are known as a parent class or super class.
- **Child / Sub class:** The class that are newly created are known as child or sub class. The sub class inherits all the properties from parent class except constructor.
- Syntax: *class childname extends parentname*

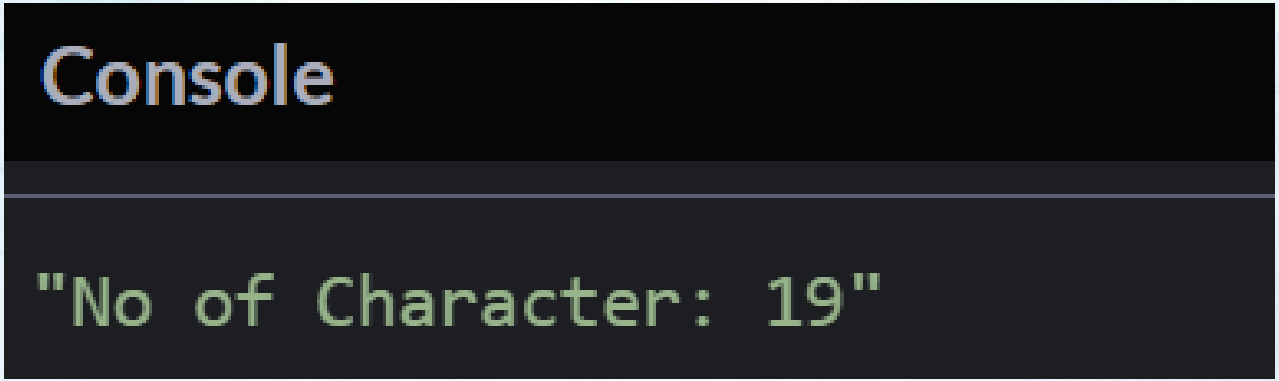
1.1.2 Fundamentals of JavaScript

Class Inheritance--Example

```
class parent {
constructor(g) {
  this.Character = g
}
}
class child extends parent {
disp() {
  console.log("No of Character: "+this.Character)
}
}
var obj = new child(19);
obj.disp()
```

1.1.2 Fundamentals of JavaScript

Class Inheritance--Example



1.1.2 Fundamentals of JavaScript

Class Inheritance—Types

Inheritance is divided into three types:

- **Single Inheritance:** Every class can be extend from one parent class.
- **Multiple Inheritance:** A class can inherit from multiple classes. ES6 does not support multiple inheritance.
- **Multilevel Inheritance:** A class can inherit from another class (via) which inherit from the parent class.
- Example:

  class Child extends Root

  class Leaf extends Child

So class Leaf indirectly extends class Root

1.1.2 Fundamentals of JavaScript

Super Keyword

- The “super” keyword helps child class to invoke the parent class data.
- The “super” keyword is used to access and call functions on an object’s parent.
- The “super” keyword is used to call the constructor of its parent class to access the parent’s properties and methods.
- Syntax:

  super.object

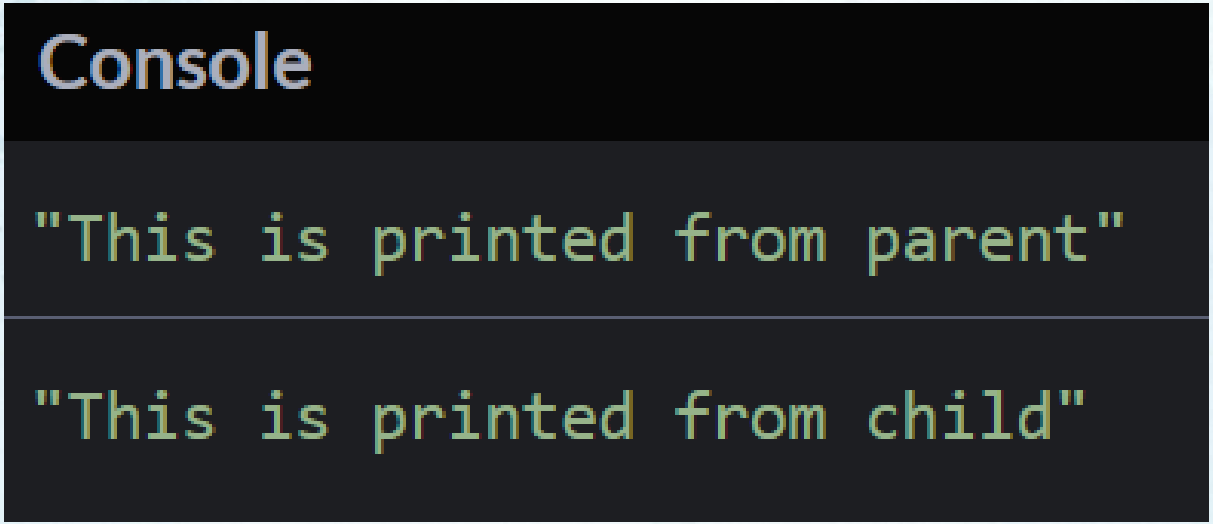
1.1.2 Fundamentals of JavaScript

Super Keyword - Example

```
//parent class
class Parent {
doPrint() {
  console.log("This is printed from parent")
}
}
//child class
class Child extends Parent {
doPrint() {
  //super keyword is used to access the function of the parent class
  super.doPrint()
  console.log("This is printed from child")
}
}
var obj = new Child()
obj.doPrint()
```

1.1.2 Fundamentals of JavaScript

Super Keyword - Example



1.2 React JS

1.2.1 What is React JS?

- React JS is basically a JavaScript library.
- React is an efficient, declarative, and flexible open-source JavaScript library for building simple, fast, and scalable frontends of web applications.
- Industry giants like Apple, Netflix, PayPal, and many others have also already started using React JS in their software productions.

1.2.2 Key Benefits of React JS

Speed:

- React basically allows developers to utilize individual parts of their application on both the client-side and the server-side.
- Which ultimately boosts the speed of the development process.

Flexibility:

- Compared to other frontend frameworks, the React code is easier to maintain and is flexible due to its modular structure.
- This flexibility, in turn, saves a huge amount of time and cost for businesses.

1.2.2 Key Benefits of React JS

Performance:

- React JS was designed to provide high performance in mind.
- The core of the framework offers a virtual DOM program and server-side rendering, which makes complex apps run extremely fast.

Usability:

- Deploying React is fairly easy to accomplish if you have some basic knowledge of JavaScript.
- In fact, an expert JavaScript developer can easily learn all ins and outs of the React framework in a matter of a day or two.

1.2.2 Key Benefits of React JS

Reusable Components:

- One of the main benefits of using React JS is its potential to reuse components.
- It saves time for developers as they don't have to write various codes for the same features.
- Furthermore, if any changes are made in any particular part, it will not affect other parts of the application.

React can be used in web development as well as mobile applications for both Android and iOS platforms.

1.2.3 Rendering Elements in React

- Elements are the smallest building block of react app.
- An element describes what you want to see on the screen.
- React elements are plain objects that are easy to create.
- Example:

```
const element = <h1>Hello, world</h1>;
```

1.2.3 Rendering Elements in React

Rendering Elements in React – Example

- Let's see how to render the element in the HTML element.
- Assume a section in a HTML file with id.
- Applications built with just React usually have a single root DOM node.
- To render a React element, first pass the DOM element to ReactDOM.createRoot() , then pass the React element to root.render().

```
<div id="root"></div>
```

1.2.3 Rendering Elements in React

Rendering Elements in React – Example

```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);
const element = <h1>Hello, world</h1>;
root.render(element);
```

1.2.4 React Components

- Components let you split the UI into independent, reusable pieces of code.
- Components are like JavaScript functions.
- Components accept arbitrary inputs called “props” and return elements describing what should be displayed on the screen.
- Types of Components:
    - Function Components.
    - Class Components.

1.2.4 React Components

Function Components

- The simplest way to define a function component is to write a JavaScript function.
- The function accepts a single “props” as an input and returns the element that describes what should be displayed on the screen.
- So the above JavaScript function is a valid React function component.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

1.2.4 React Components

Class Components

- The simplest way to define a class component is to write an ES6 class.
- When creating a React component, the component's name must start with an upper case letter.
- The component has to include the extends *React.Component* statement, this statement creates an inheritance to *React.Component*, and gives your component access to *React.Component*'s functions.
- The component also requires a *render()* method, this method returns HTML.

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

1.2.5 React Props

- Props are arguments passed into React components.
- Props are passed to components via HTML attributes.
- Props stand for properties.
- React Props are like function arguments in JavaScript *and* attributes in HTML.

```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}
```

```
const myElement = <Car brand="Ford" />;
```

I am a Ford!

1.2.6 React State

- The state is a built-in React object that is used to contain data or information about the component.
- A component's state can change over time; whenever it changes, the component re-renders.
- The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.
- The state object should be initialized in the constructor method.

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {brand: "Ford"};
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

1.2.6 React State

- The state can have many properties.
- Using the state object the property name can be rendered with the syntax: *this.state.propertyname*.

```
this.state = {
  brand: "Ford",
  model: "Mustang",
  color: "red",
  year: 1964
};
```

```
<h1>My {this.state.brand}</h1>
```

1.2.7 React Component Lifecycle

Components Lifecycle

- Each component in React has a lifecycle that helps to monitor and manipulate during its three main phases.
- The three phases are Mounting, Updating, and Unmounting.
  - **Mounting:** Putting elements in the DOM.
  - **Updating:** The component is updated whenever there is a change in the component's state or props.
  - **Unmounting:** The component is removed from the DOM.

1.2.8 React Forms

- Just like HTML forms, React uses forms to interact with the web page.
- Forms can be added like adding elements.
- In react, form data is handled by components.
- All the data is stored in the component state.

```
function MyForm() {
  return (
    <form>
      <label>Enter your name:
        <input type="text" />
      </label>
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

1.2.9 Component Interaction

There are two basic cases of communication between react components.

1. **Parent to Child communication:** Passing the props to the child components.
2. **Child to Parent communication:** Sending data back to the parent.

1.3 Assignment

1.3 Assignment

This is a table that has the details of top employees in all departments of a company. Your mission is to print the details of the employee from the **HR department** with help of JavaScript **classes**.

Name	Department
John	Marketing
Jack	Production
Smith	Electrical
Tom	HR
Jerry	Quality Control