TRAINOSYS

REACTJS & TYPESCRIPT

Day 1

HYBRID
TECHNOLOGY
COURSES

TRAINOSYS

Introduction

Introduction

- Name / Nickname
- Current Job / Position
- Tech Stack
- Choose an animal that best represents you ☺

TRAINOSYS

Jem Laguda

TRAINOSYS

Course Overview

- Introduction to ReactJS and TypeScript (Day 1)
- React Component Development (Day 2)
- React Routing and Data Fetching (Day 3)
- State Management with Redux and TypeScript (Day 4)
- Advanced Topics and Project Development (Day 5)

Program

Day 01

- Introduction to ReactJS and it's benefits
- Setting up the development environment with NodeJS and npm
- Creating a new React project with Typescript using Create React App
- Understanding the basics of TypeScript: types, interfaces, and modules
- Building a simple React component with TypeScript

Day 02

- Recap of React component structure and lifecycle methods
- Creating functional components with TypeScript
- Working with props and prop types in TypeScript
- State management in React components using hooks with TypeScript
- Building a simple React component with TypeScript
- Handling events and form inputs in TypeScript-based components

TRAINOSYS

REACTJS

HYBRID
TECHNOLOGY
COURSES

Example Projects

- https://nexus-league-of-legends.netlify.app
- https://miguelitosfoodhub.netlify.app
- https://tax-calculator-philippines-project.netlify.app

TRAINOSYS

Introduction to React

- What is React Library?
- What React does?
- What are the benefits of React?
- How to get started with React?

TRAINOSYS

What is React?

- React is a *library* for building user interfaces
- Lets you write HTML elements inside JavaScript files (and more)
- React was initially developed by Facebook around 2011

TRAINOSYS

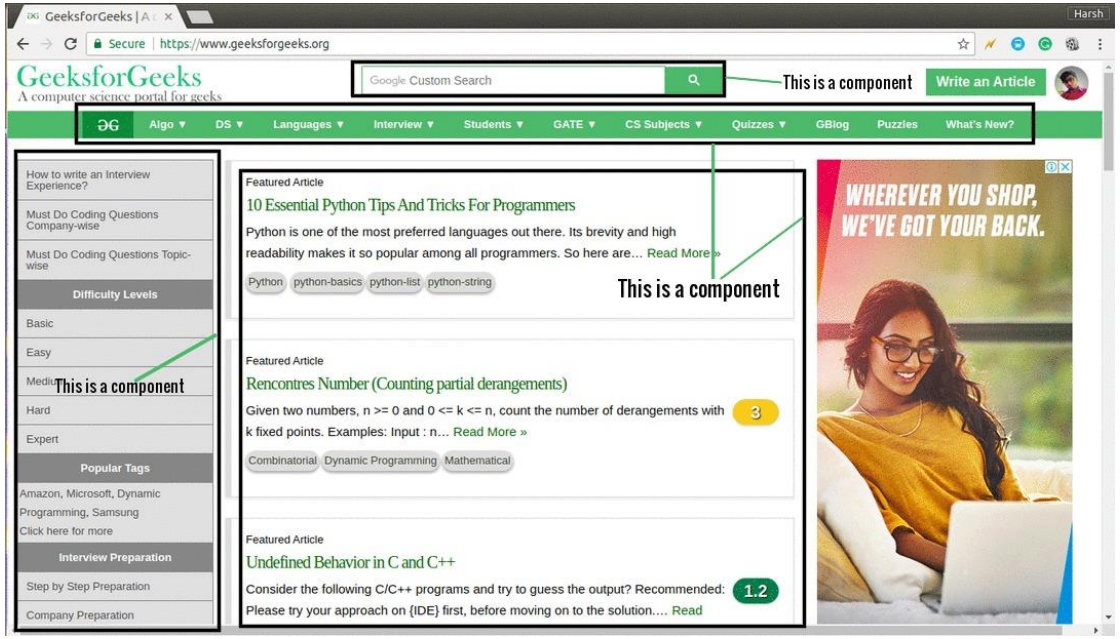
What is a library?

- In software, a 'library' is a published, self-contained set of code that you can import to help write your programs. Usually a library focuses on doing one thing well (React does UI)
- A *library* is a collection of related modules or packages
- From here on out you'll use libraries often. With JavaScript, we'll use the npm command to find and install libraries

TRAINOSYS

Why React?

Component based approach which helps in building reusable UI components (each UI works in isolation)

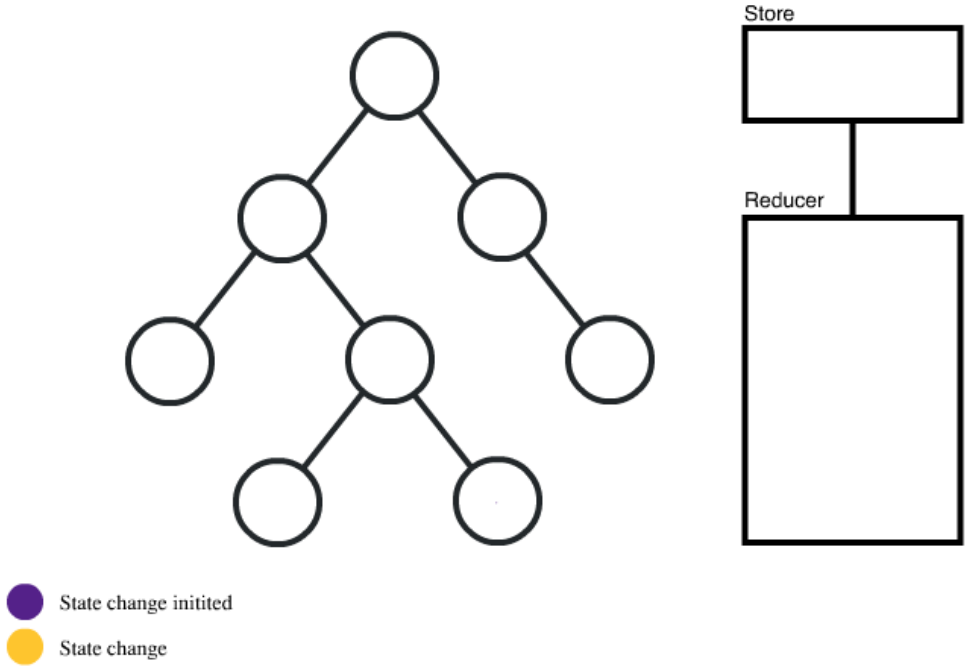


Credits to Geeks for Geeks

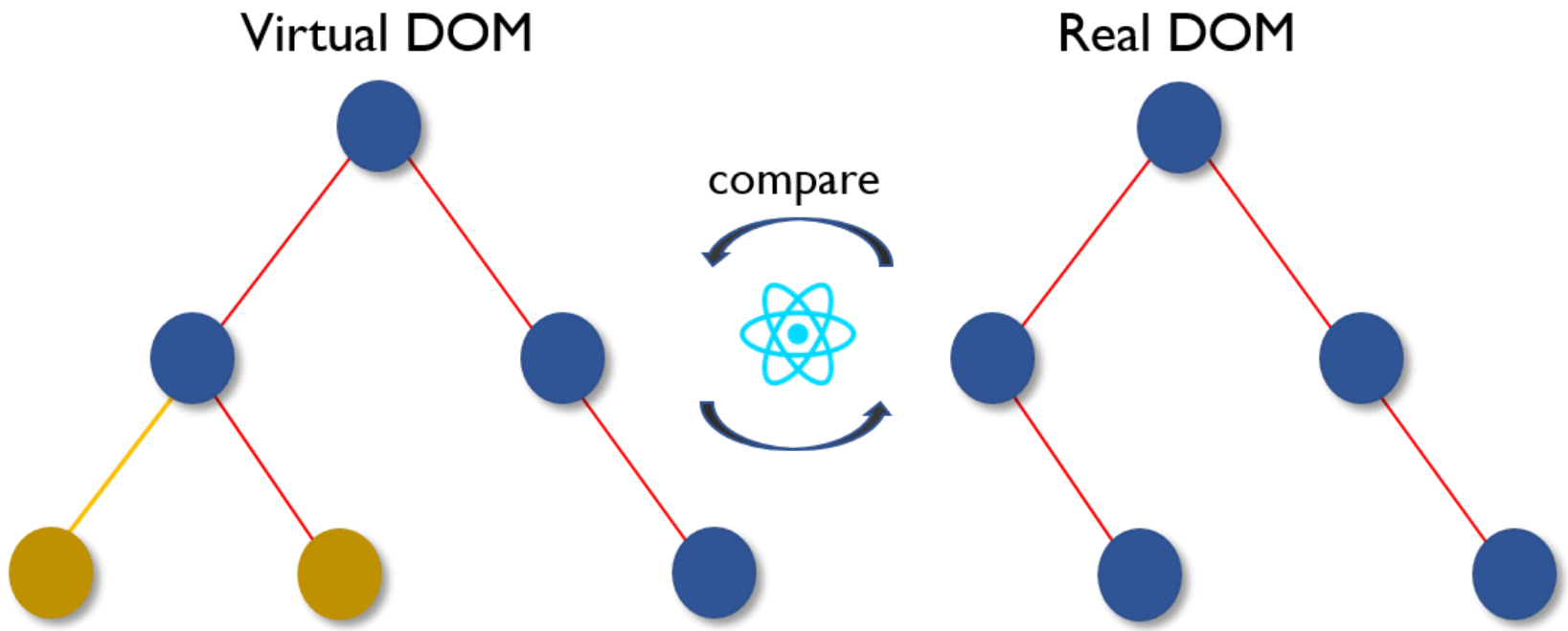
SPA - Single Page Applications (React Router Library)

https://nexus-league-of-legends.netlify.app

Unidirectional Flow



Virtual DOM



TRAINOSYS

Why React?

- Open source with robust community
- React has its own way of doing things like managing state, rendering elements and defining components

React Example Code

```
import * as React from "react";

class HelloWorld extends React.Component {
  render() {
    return (
     <div>
       <h1>Hello Delegates</h1>
       <p>And good morning!</p>
     </div>
    );
  }
}
```

React Example Code

```
import * as React from "react";

class HelloWorld extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello Delegates</h1>
        <p>And good morning!</p>
      </div>
    );
  }
}
```

imports the React Library
* (imports all)

New Class Component called "Hello World"
that extends React.Component

responsible for returning JSX
JSX - JavaScript XML

JSX - defining our HTML

TRAINOSYS

JSX vs Vanilla JavaScript

JSX

```
return (<div>Hi</div>);
```

Vanilla JavaScript

```
const div=document.createElement("div");
div.textContent = "Hi";
return div;
```

TRAINOSYS

Class-based Components

```
import React, { Component } from 'react';

class ExampleComponent extends Component {
  render() {
    return (
      <div>This is an example component.</div>
    );
  }
}

export default ExampleComponent;
```

TRAINOSYS

Functional Components

```
1   import React, { Component} from 'react';
2
3   class ExampleComponent extends Component {
4
5       render(){
6           return  <div > Hi! I'm a component!</div>
7       }
8   }
9
10  export default ExampleComponent
```

TRAINOSYS

Class-based Components vs Functional Components

```jsx
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Count updated');
  }, [count]);

  const handleClick = () => {
    setCount(prevCount => prevCount + 1);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
}
```

```jsx
import React, { Component } from 'react';

class Example extends Component {
  state = {
    count: 0
  };

  componentDidUpdate(prevProps, prevState) {
    if (prevState.count !== this.state.count) {
      console.log('Count updated');
    }
  }

  handleClick = () => {
    this.setState(prevState => ({ count: prevState.count + 1 }));
  };

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={this.handleClick}>Click me</button>
      </div>
    );
  }
}
```

Class based Components

- Extending the React.Component class from React.
- They are defined using ES6 classes and utilize the render() method to define the component's UI.
- State management is achieved using the state object, and state updates are performed using the setState() method.
- Lifecycle methods such as componentDidMount(), componentDidUpdate(), etc., are available for implementing component lifecycle behavior.
- They can hold and manage their own internal state.
- Class-based components can have their own instance methods.
- They can use the this keyword to access props and state within the component.
- Class-based components provide a way to handle complex logic and maintain local component state.

Functional Components

- Functional components are defined as JavaScript functions.
- They are simpler and more lightweight compared to class-based components.
- They receive props as the input arguments and return JSX to define the component's UI.
- Functional components are stateless by default, but the introduction of hooks in React allows them to manage state using the useState hook.
- They don't have lifecycle methods directly. However, with the introduction of hooks, functional components can use hooks like useEffect to achieve similar behavior.
- Functional components are typically easier to read, understand, and test.
- They are more focused on presenting UI based on the received props and don't have their own internal state or instance methods.

React Alternatives

- Angular Framework
- VueJS Framework
- Backbone Framework (but its not a UI Library)

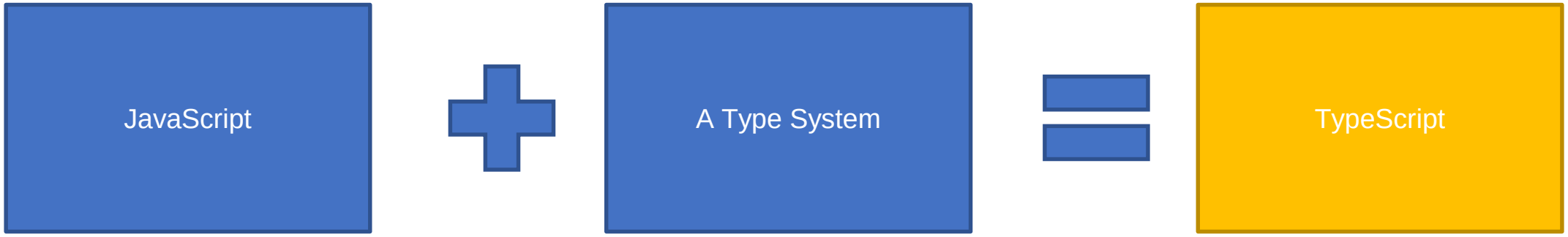
TRAINOSYS

TYPESCRIPT

HYBRID
TECHNOLOGY
COURSES

TRAINOSYS

So.. What is TypeScript?



TRAINOSYS

The TypeScript System

- Helps us catch errors during development
- Uses 'type annotations' to analyze our code
- Only active during development
- Doesn't provide any performance optimization

TRAINOSYS

TypeScript vs Vanilla Javascript

```javascript
// JavaScript code
function greet(name) {
  console.log(`Hello, ${name}!`);
}

greet("John");
```

```typescript
// TypeScript code
function greet(name: string): void {
  console.log(`Hello, ${name}!`);
}

greet("John");
```

TypeScript

- Is like a friend that helps you catch error line by line

Types

- Represents the value type or object structure
- Explicitly declared or inferred
- Type annotations ensure expected value types
- Complex types include arrays, objects, and functions
- Static typing catches type errors during development

TRAINOSYS

Example of Types

```
7
8   let age: number = 25; // declaring age as a number
9   let firstName: string = "John"; // declaring firstName as a string
10  let isActive: boolean = true; // declaring isActive as a boolean
11
```

Example of Types

```
12  // declaring numbers as an array of numbers
13  let numbers: number[] = [1, 2, 3, 4, 5];
14
15  // declaring person as an object with properties name (string) and age (number)
16  let person: { name: string, age: number } = {
17    name: "John",
18    age: 25
19  };
20
21  // declaring greet as a function that takes a string parameter and returns void
22  let greet: (name: string) => void = (name) => {
23    console.log(`Hello, ${name}!`);
24  };
25
```

Interfaces

- Define the properties and methods that the implementing objects must have
- The properties in an interface can have their own types, such as string, number, or even other interfaces
- Interfaces can be used to describe the shape of functions as well, specifying the parameter types and return type
- Purely a development-time construct for type checking

TRAINOSYS

Example usage of Interface

```
8  interface Person {
9    name: string;
10   age: number;
11   sayHello: () => void;
12 }
13
14 class Student implements Person {
15   name: string;
16   age: number;
17
18   constructor(name: string, age: number) {
19     this.name = name;
20     this.age = age;
21   }
22
23   sayHello() {
24     console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);
25   }
26 }
27
28 const john: Person = new Student("John", 25);
29 john.sayHello(); // Output: Hello, my name is John and I'm 25 years old.
30
```

TRAINOSYS

React, TypeScript and Node

Get Familiar with ReactJS + TypeScript

Towards the end, we'll build a project with React + TypeScript + Node

TRAINOSYS

CODE ALONG

HYBRID
TECHNOLOGY
COURSES

TRAINOSYS

Check list of our tools

- NodeJS
- Any IDE like VS code
- Browser
- Git
- Gitlab Account

How to create a React project Create React App

TRAINOSYS

```
npx create-react-app <app-name>
```

TRAINOSYS

Activity 1

- ❑ Create a Class based Component - DONE
- ❑ Create a Functional Component - DONE
- ❑ Display the “Hello <insert name of your crush here>” in JSX - DONE
- ❑ Create 3 nested components

TRAINOSYS
Training the Future Today

Reach Us!

Visit Us

12th/F The Trade & Financial Tower Unit 1206
32nd Street & 7th Avenue Bonifacio Global City,
Taguig 1634 Philippines

Email Us

inquiry@trainosys.com

Browse Our Website

www.trainosys.com

TRAINOSYS

Training the Future Today