styled components:

It is a third party library, a very popular one.which allows us to set pre-styled components with their own scoped styles

inline styles:

```
<input type="text" onChange={goalInputChangeHandler} value={enteredValue}
style={{borderColor: !isValid ? 'red' : 'black',background: !isValid ? 'salmon' :
'transparent'}}  placeholder={isValid===false ? 'eneter valid text' : 'enter text'}/>
```

Dyanamically adding classes:

in css file:

```
.form-control.isValid input{
  background-color: rgb(233, 167, 167);
  border-color: red;
  }

.form-control.isValid label{
    color:red}
```

In js file:

```
<div className={`form-control ${!isValid ? 'isValid' : ''}`}>
<label>Course Goal</label>
<input type="text" onChange={goalInputChangeHandler} value={enteredValue}
placeholder={isValid===false ? 'eneter valid text' : 'enter text'}/>
</div>
```

1) Regular css will not have particular scope.they are accessible to all files.It might be problem when two files are using same class name.to avoid it there are two approaches.

-> use a package called styled components.

Styled components is a package that helps you build components which have certain styles attached to them.where the styles really only affect the components to which they were attached and not any other components.

```
import styled from 'styled-components';

const Button= styled.button``;
```

This is called attacked template literal.styled is object and button is method here.between back ticks we can add any number of styles we want.this will create a button with styles that we provided for it.

```
const Button= styled.button`

  font: inherit;
  padding: 0.5rem 1.5rem;
  border: 1px solid #8b005d;
  color: white;
  background: #8b005d;
  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
  cursor: pointer;


&:focus {
  outline: none;
}

&:hover,
&:active {
  background: #ac0e77;
  border-color: #ac0e77;
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
}

`;
```

No need to use class names. For pseudo selectors ‘&’ is used.’&’ is like class name here.
psuedo selectors means :
.class_name:hover{
};

We can also pass props to dynamically change styles.

```
const FormControl=styled.div`
margin: 0.5rem 0;
& label {
  font-weight: bold;
  display: block;
  margin-bottom: 0.5rem;
  color:${props =>(props.Isvalid) ?'red' : 'black'};
}

& input {
  display: block;
  width: 100%;
  border: 1px solid ${props=>(props.Isvalid) ? 'red' : 'black'};
  background: ${props =>(props.Isvalid) ? 'rgb(233, 167, 167)': 'transparent'}
  font: inherit;
  line-height: 1.5rem;
  padding: 0 0.25rem;
}
```

```
 <form onSubmit={formSubmitHandler}>
      <FormControl Isvalid={!isValid}>
        <label>Course Goal</label>
        <input type="text" onChange={goalInputChangeHandler}
value={enteredValue} placeholder={isValid===false ? 'eneter valid text' : 'enter
text'}/>
      </FormControl>
      <Button type="submit">Add Goal</Button>
    </form>
```

We can add media query to fit button in all screens.

```
const Button= styled.button`
  width:100%;
  font: inherit;
  padding: 0.5rem 1.5rem;
```

```
  border: 1px solid #8b005d;
  color: white;
  background: #8b005d;
  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
  cursor: pointer;

@media (min-width: 600px){
    width:auto;
  }
```

Here in @media (min-width:600px) width is default width.it will change width size when screen size is less than 600px. It use above style to apply. So it uses width:100% from above styles.

-> second approach is css modules.

```
import styles from './Button.module.css'

const Button = props => {
  return (
    <button type={props.type} className={styles} onClick={props.onClick}>
      {props.children}
    </button>
  );
};

export default Button;
```

Here style becomes an object. And can use properties like 'styles.button'.

css modules It takes css classes and css files and basically changes those classes names to be unique.

We can add media query like this:

```
@media (min-width:600px){
  .button{
    width:auto;
  }
}
```