Part 3.1

Briefly explain the concept of props and state in ReactJS and how they are used to manage data flow within components:

	Props (Properties)	State
Data flow	Props is data that are passed down from parent component to child component.	State is data that a component manages internally.
Mutability	Immutable: ensuring that data flow is unidirectional from parent to child.	Mutable: only change by the component itself (use setState in class components or use setter from useState in functional components)
Usage	Used for passing data and configuration from parent to child.	Used for managing dynamic data and component behavior.
Re-render	Component (child component that receive props) will re-render when it's props change.	Component will re-render when it's state change.
Scope	External to component	Component internally

Part 4.1

Describe the concept of CSS specificity and explain how it determines which styles are applied to an element. Provide examples of different selectors and their relative specificity levels:

Concept: CSS specificity is the algorithm use the set of rules browsers to determine the CSS declaration are applied to an element when multiple, potentially conflicting, rules target the same element. The specificity algorithm calculates the weight of a CSS selector to determine which rule from competing CSS declarations gets applied to an element.

How Specificity determines applied styles:

Specificity is the algorithm that calculates the Weight of a CSS declaration. This weight is determined by counting the number of ID, CLASS, and TYPE selectors used in the rule targeting an element. The specificity is represented as four-path (INLINE-ID-CLASS-TYPE). When multiple CSS rules conflict for the same element, the declaration

from the rule with hightest specificity is applied. These paths are count with order **INLINE > ID > CLASS > TYPE**, this mean **1-0-0-0 > 0-10-0-0**.

- Inline styles column: Includes inline styles that have value is 1. Otherwise, 0.
- ID column: Includes only ID selectors, such as *#my-id*. Count the number of ID selectors in the rule.
- CLASS column: Include class selectors (e.g., *.my-class*), attribute selectors (e.g., *[type="ratio"]* or *[lang|="fr"]*), and pseudo-classed (e.g., *:hover*). Count the number of selectors in the rule.
- TYPE column: Includes type selectors (e.g., div, p) and pseudo-elements (e.g., ::before, ::after), Count the number of selectors in the rule.
- No Value: The universal selector (*), combinators (+, > , ~, - descendant), and negation pseudo-class (e.g., :where(), :not()) and its parameters aren't count, but selectors inside them are counted.

Tie-Breaking rules:

- !Important: This make the rule is override any other declaration, regardless of specificity.
- Source order: If two or more selector have the exact same specificity, the rule that appears later is the CSS will be applied.

Explaination:

Selectors	INLINE	ID	CLASS	TYPE
	1	0	0	0
#my-id	0	1	0	0
.my-class	0	0	1	0
[type="text"]	0	0	1	0
:hover	0	0	1	0
::before	0	0	0	1
*	0	0	0	0
div p	0	0	0	2
div:hover	0	0	1	1
div.my-class	0	0	1	1
.myclass:hover	0	0	2	0
.my-class.another-class	0	0	2	0
#my-id p.my-class	0	1	1	1

Example:

```
<div id=”my-id” class=”my-class”>
        <p class=”another-class”>…</p>
</div>

div.my-class .another-class { /* specificity: 0,0,2,1 */
        color: red;
}

#my-id .another-class { /* specificity: 0,1,1,0 */
        color: yellow;
}

//Applied rule
color: yellow; /* because 0,1,1,0 > 0,0,2,1 */
```

Part 4.2

What is a CSS Grid? Briefly describe its advantages over Flexbox and explain how you would use it to create a responsive two-column layout with a sidebar on the left and main content on the right.

CSS Grid: The CSS grid layout module excels (like table, but more flexible). It allows divide a page (section, menu of a page) into rows and columns, and then place elements within these defined cells by size, position, and areas.

Why CSS grid layout advantages over flexbox for layout:

	CSS Grid Layout	**Flexbox**
2D layout	Manages boths rows and columns simultaneously.	Inherently one-direction (row or column).
Layout-first design	YES: Define the overall grid structure (layout) first then place items into it.	NO: Base on number of items.
Precise Item	YES: Items can be explicitly placed in specific grid cells/areas	NO: Flexbox primarily flows items.
Placement & Overlap	YES: Items can easily placement and overlap	
Gap	Grap for both rows and columns (gap rows and column may be difference)	One gap for all.
Complex layout	YES	NO

Create a responsive two-column layout with a sidebar on the left and main content on the right:

```html
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="./style.css" />
  </head>
  <body>
    <div class="page-container">
      <div class="sidebar"></div>
      <div class="main-content"></div>
    </div>
  </body>
</html>
```

```css
/* reset */
body {
  color: white;
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

/* --- Mobile first: Default Stacked Layout --- */
.page-container {
  padding: 0.25rem;
  height: 100svh; /* cheat for full screen */
}
.sidebar,
.main-content {
  padding: 1.5rem;
  border: 1px solid rgb(77, 82, 36);
}
.sidebar {
  background-color: gray;
  margin-bottom: 1rem;
}
.main-content {
  background-color: aliceblue;
}

/* --- Tablet & Desktop: Two-column grid layout --- */
@media screen and (min-width: 768px) {
  .page-container {
```

```
    display: grid; /* enable grid layout */
    grid-template-columns: 30% 70%; /* Define two column: sidebar (30%), main (70%) */
    grid-template-areas: "sidebar main"; /* Single row with two named areas */
    gap: 0.25rem;
  }
  .sidebar {
    grid-area: sidebar; /* Assign this element to the 'sidebar' grid area */
    margin-bottom: 0; /* Remove the margin used for mobile */
  }
  .main-content {
    grid-area: main;
  }
}
```

Explanation:

- Turn page-container div into a grid container. It have to children for .slidebar and .main-content
- Define two column for children.
    - First column for .sidebar
    - Second column for .main-content will take up 1fr (one “fractional unit”) for the remaining available space in the grid container.
- Define a single row with two areas name sidebar (for .sidebar) and main (for .main-content).
- Assigns the .sidebar element to the grid area named “sidebar”.
- Assigns the .main-content element to the area named “main”.
- Responsive:
    - Mobile: @media query desn’t apply. So .page-container isn’t a grid, the sidebar and main content display as stack.
    - Tablet/Desktop: @media query apply. So .page-container is a grid and display two columns.