**** All following exams please using Javascript only**

**** 1.**

/**
There is an array, each item has such format:
{firstName: 'xxx', lastName: 'xxx', customerID: 'xxx', note: 'xxx', profession: ‘xxx’}
lastName, **note** can be empty, **customerID** can only be a set of digital numbers.
profession can only have ‘student’, ‘freelancer’, ‘productOwner’, ‘engineer’ or ‘systemAnalytics’.
**/

/**

Q1. Please follow the principle (‘firstName’ + ‘lastName’ + ‘customerID’) to sort this array and print it out.

**/

Q1. Answer

```
// Sort users based on the combined value of firstName, lastName, and
customerID.
function sortUserName(users) {
 return users.sort((a, b) => {
   const aKey = a.firstName + a.lastName + a.customerID;
   const bKey = b.firstName + b.lastName + b.customerID;
   return aKey.localeCompare(bKey);
 });
}

console.log(sortUserName(users));

// Sort users based on firstName, lastName, and customerID in ascending order.
function sortUserNameByValue(users) {
 return users.sort((a, b) => {
   if (a.firstName < b.firstName) return -1;
   if (a.firstName > b.firstName) return 1;
   if (a.lastName < b.lastName) return -1;
   if (a.lastName > b.lastName) return 1;
   if (a.customerID < b.customerID) return -1;
   if (a.customerID > b.customerID) return 1;
   return 0;
 });
}

console.log(sortUserNameByValue(users));
```

/**
Q2. Please sort by 'profession' to follow the principle.
('systemAnalytics' > 'engineer' > 'productOwner' > 'freelancer' > 'student'')
**/

```
// Sort users based on the order of their profession.
const professionOrder = new Map([
 ['systemAnalytics', 1],
 ['engineer', 2],
 ['productOwner', 3],
 ['freelancer', 4],
 ['student', 5],
]);

function sortByType(users) {
 return users.sort((a, b) => {
   return (
     professionOrder.get(a.profession) - professionOrder.get(b.profession)
   );
 });
}

sortByType(users);
```

2.

/** HTML
<div class="container">
 <div class="header">5/8 外出確認表</div>
<div class="content">
 <ol class="shop-list">
 <li class="item">麵包</li>
<li class="item">短袖衣服</li>
<li class="item">飲用水</li>

```
<li class="item">帳篷</li>
</ol>
 <ul class="shop-list">
 <li class="item">暈車藥</li>
<li class="item">感冒藥</li>
<li class="item">丹木斯</li>
<li class="item">咳嗽糖漿</li>
</ul>
 </div>
 <div class="footer">以上僅共參考</div>
</div>
**/

/** CSS
.container {
 font-size: 14px;
}
.container .header {
 font-size: 18px;
}
.container .shop-list {
 list-style: none;
 margin-left: -15px;
}
.container .shop-list li.item {
 color: green;
}
.container .shop-list .item {
 /* Explain why does this color not works, and how to fix make it work on
1st list */
 color: blue;
}

 /* Write styling make every other line give background color to next one*/
```

```
.container .shop-list .item:nth-child(odd) {
    background-color: #f0f0f0;
}
```

```
 **/
```

The reason the color does not work for the `.container .shop-list .item` is due to **CSS specificity**. The rule `.container .shop-list li.item` has higher specificity compared to `.container .shop-list .item`. This means `color: green;` will override `color: blue;`.

Here's how specificity is calculated:

1. Count the number of ID selectors in the selector (A)
2. Count the number of class selectors, attribute selectors, and pseudo-classes in the selector (B)
3. Count the number of type selectors and pseudo-elements in the selector (C)

Specificity for the selectors:

- `.container .shop-list li.item` (3 classes, 1 type) → 0-3-1
- `.container .shop-list .item` (3 classes) → 0-3-0

To make 1st list items' color blue, increase its specificity. Adding `:first-child` to the `.shop-list` selector will select the first .shop-list that is the first child of its parent:

- `.container .shop-list:first-child .item` (4 classes) → 0-4-0

```css
.container .shop-list:first-child .item{
    color: blue;
}
```

3.

```
/**

Please write down a function is used to create an array of unique values.

Example:
let items = [1, 1, 1, 5, 2, 3, 4, 3, 3, 3, 3, 3, 3, 7, 8, 5, 4, 9, 0, 1,
3, 2, 6, 7, 5, 4, 4, 7, 8, 8, 0, 1, 2, 3, 1];

output: [1, 5, 2, 3, 4, 7, 8, 9, 0, 6]
**/
```

```
function getUniqueNumber (items) {
  return [...new Set(items)];
}
```

4.

/ Can you explain about Interface and Enum, and where will you be using, please make some examples. **/**

Enums are a way to define a set of named constants. They are particularly useful when you need a fixed set of related constants, such as roles, statuses, or categories.

```
enum Profession {
 Student = 'student',
 Freelancer = 'freelancer',
 ProductOwner = 'productOwner',
 Engineer = 'engineer',
 SystemAnalytics = 'systemAnalytics',
}
```

Interfaces define the structure of an object by specifying its properties and types. They are used extensively, such as in defining object structures and `PropsType` in React.

```
interface Customer {
 firstName: string;
 lastName?: string;
 customerID: string;
 note?: string;
 profession: Profession;
}

const customer: Customer = {
 firstName: '紀',
 lastName: '又',
 customerID: '2333',
 profession: Profession.Engineer,
 // Enums can also be used in runtime to assign values.
};
```

By using enums and interfaces together, well-defined data structures can be created, enhancing the readability and reliability of the code.

5.

/ Can you explain the problem with the following code, and how to fix it. **/**

```
class Count extends React.Component {
 constructor(props) {
 super(props);
 this.state = { count: 0 };
 this.handleAddCount = this.handleAddCount.bind(this);  }

 handleAddCount(){
 this.setState({ count: this.state.count + 1 });
 this.setState({ count: this.state.count + 1 });
 this.setState({ count: this.state.count + 1 });
 }
 render() {
 return (
 <div>
 <h2>{this.state.count}</h2>
 <button onClick={this.handleAddCount}>Add</button>
</div>
 );
 }
}

ReactDOM.render(
 <Count />,
 document.getElementById('root')
);
```

The problem is that `handleAddCount` attempts to increase the count three times, expecting it to equal three. However, the result is one because React batches these updates for performance.

To fix this, use the functional form of `setState` to ensure each update is based on the latest state:

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

6.

/ Please write the sample code to debounce handleOnChange (Do not use any 3th party libs other than react) **/**

```
const SearchBox = () => {

 handleOnChange = (event) => {
 // make ajax call
 };

 return <input type="search" name="p" onChange={handleOnChange} />;
};
```

```
import { useRef, useState } from 'react';

function SearchBox() {
 const [searchTerm, setSearchTerm] = useState('');
 const debounceTimeout = useRef(null);

 const handleOnChange = (event) => {
   setSearchTerm(event.target.value);

   if (debounceTimeout.current) {
     clearTimeout(debounceTimeout.current);
   }

   debounceTimeout.current = setTimeout(() => {
     console.log('make ajax call');
   }, 1000);
 };
 return (
   <input
     type="search"
     name="p"
     onChange={handleOnChange}
     value={searchTerm}
   />
 );
}

export default SearchBox;
```

React Component Overview

Browser Support:

Chrome / Edge / Firefox

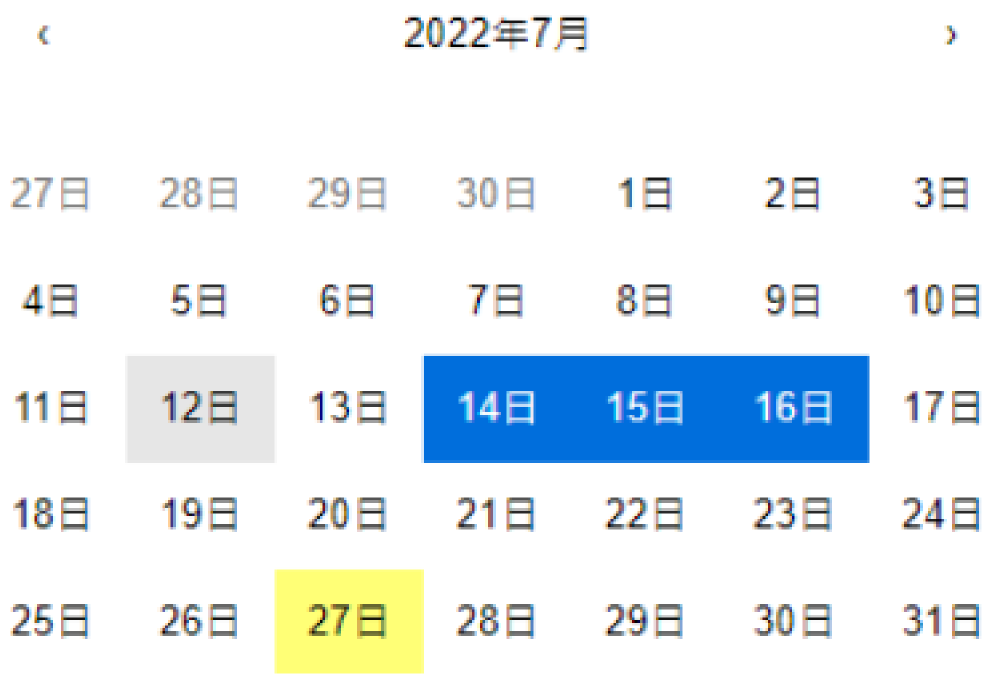
Deliverable:

1. All HTML/CSS/JS files containing the answers to the questions. (task 1 or task 2)
2. README.md containing the instruction to build and run the code.

Evaluation Criteria:

1. Feature completeness.
2. Coding style.
3. Source code structure.
4. Reusability and extensibility.

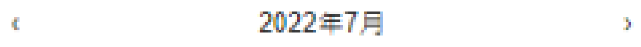
Layout


‹ 2022年7月 ›

27日	28日	29日	30日	1日	2日	3日
4日	5日	6日	7日	8日	9日	10日
11日	12日	13日	14日	15日	16日	17日
18日	19日	20日	21日	22日	23日	24日
25日	26日	27日	28日	29日	30日	31日

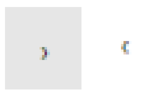
```
Width: 350px
Height: 240px
Font-size: 16px
```

Header:


‹ 2022年7月 ›

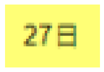
```
Width: 350px
Height: 44px
Margin-bottom: 16px
```

Month Select:



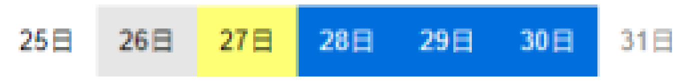
```
Width: 44px
Height: 44px
Background-color (default):
white Background-color (hover):
```
#e6e6e6

Day Button:



Width: 50px
Height: 36px

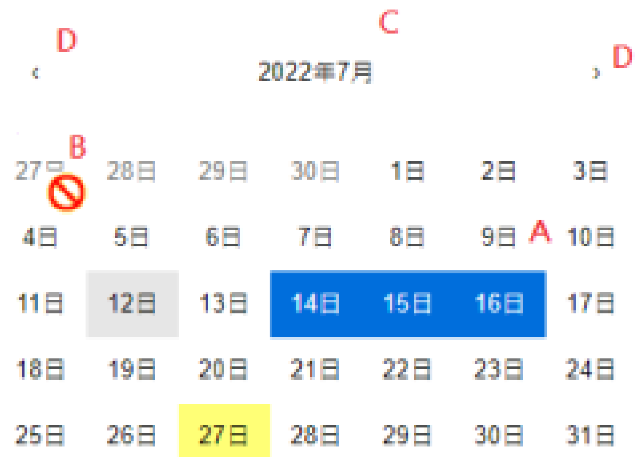
Day State:



Default: white
Hover: #e6e6e6
Today: #ffff76
Active: #006edc
Non-Current Month: #757575

Scenario (Choose one of the following tasks):

Task – 1 (Date Range Component for current month)


D
C
D
2022年7月
B
A
27日 28日 29日 30日 1日 2日 3日
4日 5日 6日 7日 8日 9日 10日
11日 12日 13日 14日 15日 16日 17日
18日 19日 20日 21日 22日 23日 24日
25日 26日 27日 28日 29日 30日 31日

A:

1. First click date to set it as start date value.
2. Next click date is same as current select option or later than current option will set it as end date value.
3. Next click date is earlier than current option will reset start date value.

B:

1. Show not-allowed icon when hovering on “Non-Current Month” day. 2. Disable day click.

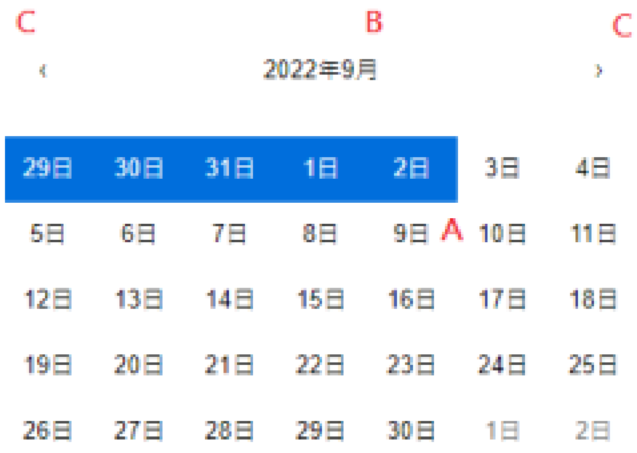
C:

1. Show current month.

D:

1. No actions.

Task – 2 (Date Range Component for cross months)



‹ 2022年9月 ›

29日	30日	31日	1日	2日	3日	4日
5日	6日	7日	8日	9日	10日	11日
12日	13日	14日	15日	16日	17日	18日
19日	20日	21日	22日	23日	24日	25日
26日	27日	28日	29日	30日	1日	2日

A:

1. First click date to set it as start date value.
2. Next click date is same as current select option or later than current option will set it as end date value.
3. Next click date is earlier than current option will reset start date value.

B:

1. Show current month.

C:

1. Click to select previous/next month.

Limination:

1. Do not use any 3th party datetime picker components.
   (ex: DateTimepicker, DatePicket etc.)
2. Allow to use date format library
   (ex: Moment.js, Date-fns).
3. Do not use any 3th party utility library.
   (ex: Lodash, Underscore etc.)
4. Bonus: Unit test.