Given the following code that uses HTML/JavaScript with DOM and Event Handling:

```
<html>
   <body>
       <input type="range" id="lengthRange" oninput="updateLength(this.value)">
       <input type="text" id="lengthText" onchange="updateLength(this.value)">
       <script>
           function getDivElement(id) {
               let el = document.getElementById(id);
               if(!el) {
                   el = document.createElement('div');
                   el.id = id;
                   document.body.appendChild(el);
               }
               return el;
           }

           function updateLength(value) {
               const el = createDivElement('cm');
               if(isNaN(value)) {
                    document.body.removeChild(el);
                    return;
               }
               document.getElementById('lengthRange').value = value;
               document.getElementById('lengthText').value = value;
               span.innerText = `${value * 10} cm`;
           }
       </script>
   </body>
</html>
```

Which of the following Vue components is equivalent to the code above and most closely follows declarative rendering with bidirectional binding?

- ```
  <template>
      <input type="range" v-model="length">
      <input type="text" v-model="length">
      {{ parseFloat(length) * 10 }} cm
  </template>
  <script>
     export default {
         data : () => {
             return {
                 length: 0
             }
        }
     }
  </script>
  ```

- ```
  <template>
      <input type="range" v-model="length">
      <input type="text" v-model="length">
      <span v-if="numInCM">{{ numInCM }} cm</span>
  </template>
  <script>
     export default {
         data : () => {
             return {
                 length: 0
             }
        },
        computed : {
            numInCM() {
                return parseFloat(this.length)*10;
            }
        }
     }
  </script>
  ```

- ```
  <template>
      <input type="range" :value="length" @input="updateValue(this.value)">
      <input type="text" :value="length" @change="updateValue(this.value)">
      {{ num * 10 }} mm
  </template>
  <script>
     export default {
         data : () => {
             return {
                 length: 0
             }
         },
         methods : {
              updateValue(value) {
                  if(!isNan(value)) {
                     this.length = parseFloat(value);
                  }
              }
         }
     }
  </script>
  ```
  ```

Consider the following Node.js API endpoint:

```
app.get('/frame/:style.:ending', (req, res) => {
    if (!req.query.quality) {
        return res.sendStatus(404);
    }
    const dir = '../resources/frame/' + req.query.quality;
    const file = `${req.params.style}.${req.params.ending}`;
    res.sendFile(path.join(__dirname, dir, file));
});
```

What happens when the following request comes in?

```
GET /frame/cedar.jpg?quality=medium
```

- Since the endpoint is designed to serve a static asset, it cannot react to any query parameters, which means the "quality" parameter will be undefined and the response is 404 Not Found.
- The endpoint wants to return a static asset using sendFile, but since the filename is constructed from dynamic parameters, it will always return 404 Not Found.
- The web server dynamically executes the endpoint to return a file based on path parameters. However, since this is a static asset, the query parameters will be undefined resulting in an Internal Server Error (500).
- The endpoint dynamically returns a file on the server based on both path and query parameters, with an HTTP response status based on parameter correctness.

Inspect the following JavaScript and HTML code

Hints:

- isFinite returns true if a passed string is a finite number
  For instance isFinite("12") returns true, isFinite("Some text") returns false
- split is a String function that takes a delimiter (e.g., ",") and turns the string into an array of strings
  For instance "1, 23, Two, 5".split(",") returns ["1", "23", "Two", "5"]

```
function sum(numberList) {
    let sum = 0;
    for(let number of numberList) {
        sum += isFinite(number) ? parseInt(number) : 0;
    }
    return sum;
}

function pointsToGrade(pointsList, gradeMap, passingFn) {
    if(!passingFn(pointsList)) {
        return "N5"
    }

    points = sum(pointsList);

    for(let grade in gradeMap) {
        if(gradeMap[grade] < points) {
            return grade;
        }
    }
    return "Not in range";
}

document.addEventListener("DOMContentLoaded", _ => {
    const pointsEl = document.getElementById('points');
    pointsEl.addEventListener('change', event => {
        const points = event.target.value;
        const pointsList = points.split(',');
        const grades = {
            "S1" : 87, "U2" : 74, "B3" : 62,
            "G4" : 50
        }
        document.getElementById('output').innerText =
            pointsToGrade(pointsList,
                          grades,
                          x => sum(x) > 50
            );
    });
});
```

```
<form>
    <input type="text" id="points" />
</form>
<div id="output"></div>
```

Which of the following answers is true or false?

True	False	
○	○	Passing in a mixed list of numbers and strings (e.g., "asdf, 25, 25, 20" or "20, something, 23") always results in an output of "N5"
○	○	The input "Twenty, 25, 25, 20" returns "B3"
○	○	The input "30, 21" returns "G4"
○	○	Any list of points that sums to less than 50 results in an output of "Not in range"

Given this markup

```
<div style="border: 4px solid black;">
  <div id="outer">
    <span id="inner" style="font-size: 2em;">Some text</span>
  </div>
</div>
```

Which CSS declaration is most likely to result in the following rendering in the browser?

- `#outer { margin: 2em; border: 4px solid red; } #inner { padding: 5em; }`
- `#outer { padding: 5em; margin: 2em; border: 4px solid red; }`
- `#outer { padding: 2em; } #inner { margin: 5em; border: 4px solid red; }`
- `#outer { margin: 2em; padding: 1em; } #inner { border: 4px solid red; margin: 2em; padding: 1em;}`

Given this CSS code:

```
p {
  --info-color: lightblue;
}

p.info {
  font-size: 3em;
  color: var(--info-color, blue);
}

@media only screen and (min-width: 500px) {
  p.info {
    font-size: 5em;
    color: navy;
  }
}
```

The element ... has the following styling on a device with 499px width:

font-size:

color:

A **WebSocket** connection allows

- ○ bidirectional
- ○ stateless
- ○ idempotent
- ○ RESTful

communication between a

- ○ proxy
- ○ API
- ○ client
- ○ router

and a

- ○ database
- ○ browser
- ○ server
- ○ cache

A WebSocket connection starts with

- ○ Handshake
- ○ tunneling
- ○ token exchange
- ○ URL redirect

and then switches to the

- ○ FTP protocol
- ○ WebSocket protocol
- ○ REST
- ○ HTTPS protocol

for communication. WebSockets are particularly useful for

- ○ static files
- ○ authentication
- ○ real-time
- ○ batch

applications such as

- ○ chat

- ○ login
- ○ upload
- ○ rendering

You are writing a Vue component named "AddressBook" that is supposed to display a list of names and addresses. When the component is instantiated, the data for all address book items has to be retrieved from the backend.

Which of the following answers proposes the best solution given what you have learned in the lecture?

- ○ You cannot load data from the backend within the component. The list of the data has to be passed as a property to the instance.
- ○ You have to create a function in the "methods" block of the component that fetches the data, which you can then call in the template.
- ○ We load the data in the created() hook, so that Vue can inject the data before the DOM is loaded.
- ○ We load the data in the mounted() hook, so that Vue can inject the data in the already loaded DOM.

You are reviewing code for an HTML form. While the markup looks the way it should in the browser, you can see how the following code could create accessibility issues:

```
<form action="/visa">
  <img src="instructions.png">
  <i>Visa Application</i><br>
  <table>
     <tr>
       <td>
         Passport Number<br>
         Full Name
       </td>
       <td>
         <input name="field_1"><br>
         <input name="field_2"><br>
         <input type="submit" value="Create application">
       </td>
     </tr>
  </table>
</form>
```

Which of the following actions would improve the accessibility of this HTML form and which would result in no change to accessibility?

Improved Accessibility	No change	
○	◉	Submitting the form with JavaScript, instead of using the action attribute
◉	○	Replacing Visa Application with Visa Application
◉	○	Adding labels for inputs
◉	○	Adding alternative text to the image

There are a number of errors in the following Vue component:

```
<template>
    <label>From <input type="text" v-model="range.min" @input="validate()"></label>
    <label>To <input type="text" v-model="range.max" @input="validate()"></label>
    {{ randomInRange }}
  </template>
  <script>
  export default {
    name: 'RandomNumber',
    props: {
      range: {
        min: Number,
        max: Number
      }
    },
    data() {
      return {
        validate() {
          if (range.min > range.max) {
            alert('From must be less than To');
          }
        }
      }
    },
    computed: {
      randomInRange() {
        let x = Math.random() * (this.range.max - this.range.min + 1);
        return Math.floor(x) + this.range.min;
      }
    }
  }
 </script>
```

Which of the following statements are true?

True	False	
○	○	A computed property can not be used directly in the template, it must be called as a function.
○	○	validate() is a function and needs to be part of methods, not data.
○	○	Range needs to be part of data, not props, because it is used in a two-way binding.
○	○	There can be no alert call in an input validation function.

A **session** cookie stores some kind of identifier that [ ] can map to a unique session and related data. It is first sent by the server and is subsequently sent with every [ ] from [ ] to [ ].

A **session cookie** stores some kind of identifier that

- ○ the client
- ○ the server
- ○ the browser

can map to a unique session and related data. It is first sent by the server and is subsequently sent with every

- ○ redirect
- ○ request
- ○ response

from

- ○ the server
- ○ the browser or the server
- ○ the browser

to

- ○ the browser
- ○ the server
- ○ the browser or the server

Which of the following CSS selectors would you use to target the HTML element (and only that element) highlighted in the selection below?

```
<article id="first" class="content">
    <h1>Style</h1>
    <section class="wrapper" id="wrapper">
        <ul class="itemize">
            <li>First</li>
            <li>Second</li>
        </ul>
        <div>
            <ul class="itemize">
                <li>First</li>
                <li>Second</li>
            </ul>
        </div>
    </section>
    <ul class="itemize">
        <li>First</li>
        <li>Second</li>
    </ul>
    <section>
        <p class="content">Another Style</p>
        <ul class="itemize">
            <li>First</li>
            <li>Second</li>
        </ul>
        <div>
            <ul class="itemize">
                <li>First</li>
                <li>Second</li>
            </ul>
        </div>
    </section>
</article>
```

- `article#first.content > section#wrapper ul.itemize`
- `article#first.content > section#wrapper + ul.itemize`
- `article#first.content > h1 + section.wrapper > ul.itemize`
- `article#first.content + h1 > section#wrapper + ul.itemize`

Your web shop backend connects to an external payment provider. For development, you use a sandboxed test account. When deploying your application to production, you connect to the payment provider using the actual production API keys.

Which concept we have heard about in the lecture would you use to manage these external web service credentials within your code?

- Environment variables
- Templating
- Query Parameters
- Cookies

You have a few physical servers distributed in data centers around the world. Each physical server can handle many simultaneous connections. To minimize latency, you want every user of your application to be connected to the nearest server.

Which of the following concepts you heard about in the lecture is the closest related to this description?

- Geo DNS
- Platform-as-a-Service (PaaS)
- Infrastructure-as-a-Service (IaaS)
- Virtual Machines

In the following code, how would you declare the variables note, begin, and timePassed? For each of the three variables, choose the most restrictive scope possible.

```
function stopwatch(f) {
  [ ]  note = '';
  [ ]  begin = Date.now();
  f();
  [ ]  timePassed = Date.now() - begin;
  if (timePassed > 5000) {
     note = `Wow, this function took ${timePassed/1000} seconds to run!`;
  } else {
      note = `The function ran for ${timePassed/1000} seconds.`;
  }
  return note;
}
```

A blogging platform exposes the following API, which unfortunately violates some REST principles:

```
GET /users/{id}?blogPostId={postId}
POST /users/{id}?method=addBlogPost
POST /users/{id}?method=editBlogPost&blogPostId={postId}
POST /users/{id}?method=deleteBlogPost&blogPostId={postId}
```

You ask four colleagues to re-design the endpoints so that they are more RESTful. Which of them got it correct the most?

-
```
GET /getBlogpost?userId={userId}&postId={postId}
POST /addBlogpost?userId={userId}&postId={postId}
POST /editBlogpost?userId={userId}&postId={postId}
DELETE /deleteBlogpost?userId={userId}&postId={postId}
```

-
```
GET /users/{id}/posts/{postId}
POST /users/{id}/posts
PUT /users/{id}/posts/{postId}
DELETE /users/{id}/posts/{postId}
```

-
```
GET /getBlogpost?userId={userId}&postId={postId}
POST /addBlogpost?userId={userId}&postId={postId}
PUT /editBlogpost?userId={userId}&postId={postId}
DELETE /deleteBlogpost?userId={userId}&postId={postId}
```

-
```
GET /users/{id}/posts/?postId={postId}
POST /users/{id}/posts
PUT /users/{id}/posts?postId={postId}
DELETE /users/{id}/posts/?postId={postId}
```