Sample solution: accessible font size

First of all, we can use aria-role to group the two buttons into a group whose name is identified by the aria-label. Since the texts "A+" and "A-" do not convey clearly the purposes of the buttons, we can further use aria-label so that the purposes of the two buttons will be pronounced by screen reader as "Increase font size" and "Decrease font size" respectively.

```
<div class="btn-group btn-group-sm" role="group" aria-label="Font size">
    <button class="btn btn-dark font-increase-button" id="font-increase-button"
            aria-label="Increase font size">A+</button>
    <button class="btn btn-dark font-decrease-button" id="font-decrease-button"
            aria-label="Decrease font size">A-</button>
</div>
```

*The idea behind dynamically changing the font size is quite straightforward. Note that all font sizes in the sample Website are defined using the **rem** unit, where the term rem stands for the "root element" and 1 rem is equal to the computed value of font-size on the root element (or the html element). The following snippet illustrates how rem can be used:*

```
html { font-size: 10px; }  /* =10px */
body { font-size: 1.4rem; } /* =14px */
h1   { font-size: 2.4rem; } /* =24px */
```

What we need to do is to use JavaScript to listen click events on the increase/decrease font size buttons, and then update the font-size of the root html element.

```
document.addEventListener('DOMContentLoaded', function() {
    // Add event listeners
    document
        .getElementById('font-increase-button')
        .addEventListener('click', function(event) {
            updateFontSize(event, true);
        }, false);

    document
        .getElementById('font-decrease-button')
        .addEventListener('click', function(event) {
            updateFontSize(event, false);
        }, false);
}, false);
```

```
/**
 * Increase or decrease the font size
 * @param {object} event - The DOM event
 * @param {boolean} increase - Whether to increase or decrease the font size
 */
function updateFontSize(event, increase) {
    var html = document.getElementById('root');
    var size = parseFloat(html.style['font-size']);
    if (isNaN(size)) {
        size = 1.0;
    }

    if (increase && size < 3.0) {
        size += 0.1;
    } else if (!increase && size > 0.5) {
        size -= 0.1;
    }

    html.style['font-size'] = size + 'rem';
}
```

Sample solution: drop-down menu & menu keyboard interaction

How can we make the screen reader become aware of whether there exists sub-menu of not? The answer is simply to set the aria-haspopup attribute to true on menu items that have sub-menu.

How can we notify the screen reader when a sub-menu is expanded or collapsed? The solution is to use update the aria-expanded attribute dynamically to true or false using JavaScript.

How can we make the screen reader to treat the anchor links like buttons? This can be done by setting the role of the link to button.

Lastly, if we want to collapse the menu when the ESC key is pressed, we can add a even listener on keyup events for each menu item and close the menu item if the key is pressed.

```
<ul class="navbar-nav mr-auto">
    <li class="nav-item">
        <a class="nav-link" href="./">Homepage</a>
    </li>
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="nav-bar-faculties"
           role="button" aria-haspopup="true" aria-expanded="false">
            Faculties
        </a>
        <ul class="dropdown-menu" aria-labelledby="nav-bar-faculties">
            <li>
                <a class="dropdown-item"
                   href="./empty.html?title=Applied+sciences">
                    Applied sciences
                </a>
            </li>
            <li>
                <a class="dropdown-item"
                   href="./empty.html?title=Architecture+%26+planning">
                    Architecture &amp;amp; planning
                </a>
            </li>
            …
        </ul>
    </li>
</ul>
```

```
document.addEventListener('DOMContentLoaded', function() {
    var dropDownToggles =
        document.querySelectorAll('.nav-item .dropdown-toggle');

    for (var i = 0; i < dropDownToggles.length; i++) {
        dropDownToggles[i].addEventListener('click', openMenu, false);
        dropDownToggles[i].addEventListener('keyup', closeMenu, false);
    }
}, false);
```

```
/**
 * Open the current clicked menu and close the other menus
 * @param {object} event - The DOM event
 */
function openMenu(event) {
    event.stopPropagation();
    event.preventDefault();

    var currentDropDownButton = event.target;
    var currentDropDownMenu =
        currentDropDownButton.parentNode.querySelector('.dropdown-menu');
    var isOpen = currentDropDownMenu.classList.contains('show');
    var dropdownButtons =
        document.querySelectorAll('#nav-bar-content .dropdown .dropdown-toggle');
    var dropDownMenus =
        document.querySelectorAll('#nav-bar-content .dropdown .dropdown-menu');
    for (var j = 0; j < dropDownMenus.length; j++) {
        dropDownMenus[j].classList.remove('show');
        dropdownButtons[j].setAttribute('aria-expanded', false);
    }

    if (!isOpen) {
        currentDropDownMenu.classList.add('show');
        currentDropDownButton.setAttribute('aria-expanded', true);
    }
}
```

```
/**
 * Close the current menu when ESC key is pressed
 * @param {object} event - The DOM event
 */
function closeMenu(event) {
    if (event.keyCode !== 27) {
        return;
    }

    event.stopPropagation();
    event.preventDefault();

    var currentDropDownButton = event.target;
    var currentDropDownMenu =
        currentDropDownButton.parentNode.querySelector('.dropdown-menu');

    currentDropDownButton.setAttribute('aria-expanded', false);
    currentDropDownMenu.classList.remove('show');
}
```

```
.dropdown-menu { display: none; }
.dropdown-menu.show { display: block; }
```

Sample solution: form input validation

Using the login form as an example, what we can improve in terms of accessibility are the following aspects:

1. *Add detailed error message.*
2. *Add aria live regions to notify screen reader users about the existence of mistakes.*
3. *Use anchor link to point the error message to the corresponding input control.*
4. *To further improve the visual appearance, we can provide redundant error messages:*
    a. *The first group are intended for screen readers. Therefore, they are visually hidden to normal users by setting visibility to hidden.*
    b. *The second group are intended for normal users and they are styled nicely using CSS. But they are hidden from screen readers by setting aria-hidden to true.*

```
<div class="p-4 mr-md-3 bg-white">
    <h1 class="text-primary mb-4">Login</h1>
    <!- Aria live region -->
    <h2 class="text-danger mb-4 d-none" id="login-error"
        aria-live="assertive">Please correct the error(s)</h2>
    <div class="sr-only">
        <!-- The errors here are only shown to screen readers -->
        <ul id="login-error-list"></ul>
    </div>
    <form id="login-form" novalidate>
        <div class="form-group">
            <label for="login-email-control">Email</label>
            <input class="form-control" id="login-email-control" type="email"
                   placeholder="Email" required aria-required="true">
            <!-- The error here is only shown to normal users -->
            <small class="invalid-feedback" aria-hidden="true"></small>
        </div>
        …
    </form>
</div>
```

```
.d-none { display: none; } /* Display none */
.sr-only { visibility: hidden } /* Visibility hidden */
```

```
document.addEventListener('DOMContentLoaded', function() {
    // Add event listener
    document
        .getElementById('login-login-button')
        .addEventListener('click', login, false);
```

```
/**
 * Validate the login form and try to log the user in
 * @param {object} event - The DOM event
 */
function login(event) {
    event.preventDefault();
    event.stopPropagation();

    var hasError = false;
    var error = document.getElementById('login-error');
    var errorList = document.getElementById('login-error-list');
    clearErrorList(errorList);

    var email = document.getElementById('login-email-control');
    if (email.validity.valid) {
        setValid(email);
    } else if (email.validity.valueMissing) {
        var message = 'Email cannot be empty.';
        setInvalid(email, message);
        hasError = true;
        appendErrorToList(errorList, email, message);
    } else {
        var message = 'Email is not valid.';
        setInvalid(email, message);
        hasError = true;
        appendErrorToList(errorList, email, message);
    }

    var password = document.getElementById('login-password-control');
    if (password.value.trim().length == 0) {
        var message = 'Password cannot be empty.';
        setInvalid(password, message);
        hasError = true;
        appendErrorToList(errorList, password, message);
    } else {
        setValid(password);
    }

    if (hasError) {
        error.classList.remove('d-none');
        document.querySelector('#login-error-list li a').focus();
    } else {
        error.classList.add('d-none');
    }
}
```

```
/**
 * Set the form control element to valid
 * @param {object} element - The DOM element
 */
function setValid(element) {
    element.classList.remove('is-invalid');
    element.classList.add('is-valid');
    element.parentElement.children[2].innerHTML = '';
}
```

```
/**
 * Set the form control element to invalid with the error message
 * @param {object} element - The DOM element
 * @param {object} message - The error message
 */
function setInvalid(element, message) {
    element.classList.remove('is-valid');
    element.classList.add('is-invalid');
    element.parentElement.children[2].innerHTML = message;
}
```

```
/**
 * Remove validation information from the element
 * @param {object} element - The DOM element
 */
function removeValidation(element) {
    element.classList.remove('is-valid');
    element.classList.remove('is-invalid');
    element.parentElement.children[2].innerHTML = '';
}
```

```
/**
 * Apped error to the list
 * @param {object} list - The list DOM element to append to
 * @param {object} control - The form control DOM element
 * @param {string} message - The associated error message
 */
function appendErrorToList(list, control, message) {
    var li = document.createElement('li');

    var a = document.createElement('a');
    a.innerHTML = message;
    a.href = '#' + control.id;
    a.addEventListener('click', function(event) {
        event.preventDefault();
        event.stopPropagation();
        control.focus();
    }, false);
    li.appendChild(a);
    list.appendChild(li);
}
```

```
/**
 * Remove all list items from the list
 * @param {object} list - The list DOM element to clear
 */
function clearErrorList(list) {
    while (list.firstChild) {
        list.removeChild(list.firstChild);
    }
}
```