JavaScript

Celem ćwiczenia jest przygotowanie formularza HTML z wykorzystaniem języka JavaScript. Formularz ten będzie sprawdzany pod względem zawartości przed wysłaniem do serwera. Formularz będzie miał charakter dynamiczny, tzn. niektóre jego elementy będą zmieniać swój stan pod wpływem działań użytkownika. Do wykonania ćwiczenia potrzebny jest dowolny edytor plików tekstowych oraz przeglądarka.

1. Stwórz dwa pliki tekstowe znajdujące się w tym samym katalogu dyskowym: *form.html* i *form_check.js*.
2. Formularz ma służyć do wprowadzania danych potrzebnych do rejestracji użytkownika w serwisie internetowym. Potrzebne informacje to przede wszystkim dane osobowe. W celu utworzenia formularza wykorzystaj element `<form>` języka HTML. Formularz o nazwie `dane_osobowe` powinien umożliwiać wprowadzanie następujących danych: **imię**, **nazwisko**, **płeć** (wybór jednej z opcji), **nazwisko panieńskie**, **e-mail**, **ulica**, **kod pocztowy**, **miasto**, **uwagi** (pole tekstowe). Dla lepszej wizualizacji formularza pola można umieścić w komórkach tabeli. Zawartość pliku *form.html* powinna być następująca:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>JavaScript</title>
</head>
<body>
    <form name="dane_osobowe">
        <table>
            <tr><td>Imię</td><td><input type="text" name="f_imie"></td></tr>
            <tr><td>Nazwisko</td><td><input type="text" name="f_nazwisko"></td></tr>
            <tr>
                <td>Płeć</td>
                <td>
                    <input name="f_plec" value="f_k" checked type="radio" />kobieta<br />
                    <input name="f_plec" value="f_m" type="radio" />mężczyzna
                </td>
            </tr>
            <tr><td>N. panieńskie</td><td><input type="text" name="f_nazwisko_p"></td></tr>
            <tr><td>E-mail</td><td><input type="text" name="f_email"></td></tr>
            <tr><td>Kod pocztowy</td><td><input type="text" name="f_kod"></td></tr>
            <tr><td>Ulica/Osiedle</td><td><input type="text" name="f_ulica"></td></tr>
            <tr><td>Miasto</td><td><input type="text" name="f_miasto"></td></tr>
            <tr><td>Uwagi</td><td><textarea rows="5" cols="15" name="uwagi"></textarea>
            </td></tr>
            <tr><td colspan="2"><input type="submit" value="Prześlij"></td></tr>
        </table>
    </form>
</body>
</html>
```

3. W pliku *form_check.js* zostaną umieszczone definicje funkcji, które będą sprawdzały elementarne warunki, które powinny spełniać wartości wprowadzane do formularza. Sprawdzana powinna być długość opisu uwag, format kodu pocztowego oraz adresu e-mail. Zaczniemy od napisania funkcji sprawdzającej, czy dane pole jest puste. Zadeklaruj funkcję o nazwie `isEmpty`, która przyjmie jeden parametr (będziemy do niej przekazywali ciąg znaków) i zwraca wartość `true`, jeśli ciąg znaków jest pusty, a `false` w przeciwnym wypadku. Posłuż się polem `length` przechowującym długość ciągu znaków.
4. Wykorzystamy teraz zadeklarowaną funkcję do sprawdzenia, czy użytkownik podał imię. W tym celu zadeklaruj kolejną funkcję o nazwie `validate`, do której będziemy przekazywali formularz,

który ma zostać zweryfikowany. Na razie funkcja ma wywołać funkcję `isEmpty` przekazując jej jako parametr wartość pola `f_imie` formularza (`formularz.elements["f_imie"].value`). Gdy pole okaże się puste funkcja wyświetla okienko ostrzeżenia z odpowiednim komunikatem (`alert("Podaj imię!");`) i zwraca wartość `false`. W przeciwnym wypadku – zwraca wartość `true`.

5. Dodaj do nagłówka HTML w pliku *form.html* odwołanie do zewnętrznego pliku (*form_check.js*) zawierającego skrypt w języku JavaScript.
6. Dodaj do przycisku w formularzu obsługę kliknięcia (zdarzenie `onclick`). Jako akcję na to zdarzenie wywołaj funkcję `validate` przekazując jej jako parametr formularz (`validate(this.form);`).
7. Uruchom formularz w przeglądarce i przetestuj jego działanie próbując zgłosić formularz bez podania imienia. Zwróć uwagę na zachowanie strony po zamknięciu okna ostrzeżenia. Mimo błędu formularz został wysłany!
8. Dodaj przed wywołaniem funkcji `validate` w zdarzeniu `onclick` klauzulę `return` (`return validate(this.form);`). Ponownie przetestuj działanie strony. Czy widzisz różnicę w działaniu?
9. Formularz nadal można łatwo oszukać, wpisując do pola `f_imie` ciąg białych znaków. Dodaj do pliku *form_check.js* funkcję sprawdzającą, czy podany łańcuch znaków nie składa się w całości z białych znaków lub jest pusty.

```
function isWhiteSpaceOrEmpty(str) {
    return /^[\s\t\r\n]*$/.test(str);
}
```

10. Zmodyfikuj funkcję `validate()` wykorzystując powyżej zdefiniowaną funkcję. Sprawdź, czy walidacja działa poprawnie.
11. Zmodyfikujemy teraz kod skryptu w taki sposób, aby wygodnie można było dodawać walidację poprawności kolejnych pól. Dodaj do pliku *form_check.js* funkcję `checkString`, która ma przyjmować dwa parametry: ciąg znaków do sprawdzenia oraz wiadomość do wyświetlenia w przypadku gdy ciąg okaże się pusty lub będzie składał się z samych białych znaków. Funkcja zwraca `false` w przypadku błędnego ciągu i `true` w przypadku poprawnego.
12. Zmień funkcję `validate()`. Wykorzystaj funkcję `checkString()` i dodaj sprawdzanie poprawności **nazwiska**, **kodu**, **ulicy** i **miasta**. Sprawdź działanie formularza.
13. W następnym kroku uzupełnimy skrypt o funkcję weryfikującą podstawową poprawność podanego adresu e-mail. W tym celu ponownie posłużymy się mechanizmem wyrażeń regularnych. Dodaj do pliku *form_check.js* poniższy kod.

```
function checkEmail(str) {
    let email = /^[a-zA-Z_0-9\.]+@[a-zA-Z_0-9\.]+\.[a-zA-Z][a-zA-Z]+$/;
    if (email.test(str))
        return true;
    else {
        alert("Podaj właściwy e-mail");
        return false;
    }
}
```

14. Aby uruchomić weryfikację adresu email, wprowadź konieczną modyfikację w funkcji `validate()`.
15. Przedstawione rozwiązanie nie jest zadowalające, ponieważ zmusza użytkownika do dwukrotnego kliknięcia myszką (zamknięcie okna z powiadomieniem oraz przejście do błędnie wypełnionego pola). Następny przykład pokaże, jak temu zaradzić. Zmodyfikuj formularz poprzez dodanie po polu do wprowadzania imienia pustego pola do wyświetlania błędów.

```
<span class="err" id="e_imie" />
```

16. Dołącz do strony plik ze stylami i dodaj regułę formatującą komunikat obłędzie. Reguła powinna dotyczyć wszystkich elementów `<span>` z klasą `err`.

```
color: red;
font-weight: bold;
padding-left: 5px;
```

17. Dodaj definicję nowej funkcji do weryfikacji pola.

```
function checkStringAndFocus(obj, msg) {
    let str = obj.value;
    let errorFieldName = "e_" + obj.name.substr(2, obj.name.length);
    if (isWhiteSpaceOrEmpty(str)) {
        document.getElementById(errorFieldName).innerHTML = msg;
        obj.focus();
        return false;
    }
    else {
        return true;
    }
}
```

18. Konieczna jest także modyfikacja funkcji `validate()`. Przeprowadź ten sam proces dla każdego zwalidowanego dotychczas pola w formularzu. Pamiętaj o dodaniu dodatkowych elementów wyświetlających błąd w strukturze dokumentu. Dla adresu email potrzebujesz również osobną funkcję – `checkEmailAndFocus`.
19. Niestety, tak zdefiniowany komunikat o błędzie jest wyświetlany nawet wówczas, gdy użytkownik poprawi już swój błąd, co może być mylące. Zmodyfikuj funkcje `checkStringAndFocus` oraz `checkEmailAndFocus` tak aby w przypadku braku błędów czyściły pole zawierające komunikat z błędem. Usuń również koniecznie alert z funkcji sprawdzającej poprawność adresu email.
20. Uruchom formularz i przetestuj działanie mechanizmu raportowania błędów.
21. Nie da się ukryć, że w tym momencie przydałoby się trochę porządków w naszym kodzie. Proponuję następujące rozwiązanie.
    - Połączenie funkcji `checkStringAndFocus` i `checkEmailAndFocus` w jedną (`checkStringAndFocus`) przyjmującą jako trzeci parametr funkcję walidującą.
    - W funkcji `validate`, wywoływanie jedynie powyższej funkcji dla każdego pola z odpowiednią funkcją walidującą przekazaną jako trzeci parametr (ponieważ konieczne będzie zanegowanie wartości zwracanej przez funkcję `checkEmail`, proponuję również zmianę jej nazwy np. na `isEmailInvalid`).
    - Usunięcie wszystkich pozostałych funkcji.
22. Kolejnym krokiem ćwiczenia będzie rozbudowanie formularza w taki sposób, aby pole `f_nazwisko_p` było dostępne tylko dla osoby, która jest kobietą. W tym celu konieczne będzie obsłużenie zdarzenia polegającego na przełączeniu pomiędzy wartościami pola opcji `f_plec`. W zależności od kierunku zmiany pole `f_nazwisko_p` powinno stać się widoczne lub niewidoczne. Do tego celu zostanie wykorzystana właściwość stylu o nazwie `visibility`. Może ona przyjąć wartości `visible` albo `hidden`. Zmiana nastąpi przy zajściu zdarzenia `onclick`.
23. Dodaj do pliku ze skryptami kod odpowiedzialny za zmianę właściwości `visibility` dla elementu, którego identyfikator jest przekazywany do funkcji jako argument.

```
function showElement(e) {
    document.getElementById(e).style.visibility = 'visible';
}

function hideElement(e) {
    document.getElementById(e).style.visibility = 'hidden';
}
```

24. Rozbuduj definicję pola służącego do wprowadzania nazwiska panieńskiego w taki sposób, aby możliwe było jego ukrywanie. W tym celu opakuj pole o nazwie `f_nazwisko_p` elementem `<span>` o identyfikatorze `NazwiskoPanienskie`.
25. Dodaj obsługę zdarzenia polegającego na wybraniu jednej z opcji oznaczających płeć. W zależności od wybranej płci powinno nastąpić ukrycie albo wyświetlenie pola do wprowadzenia nazwiska panieńskiego. W pierwszym elemencie `f_plec` ustaw wartość atrybutu `onclick` na odpowiednie wywołanie funkcji `showElement`, a w drugim na odpowiednie wywołanie funkcji `hideElement`.
26. Uruchom formularz w przeglądarce i sprawdź jego działanie.
27. W dotychczasowych przykładach wyszukiwaliśmy elementy według ich identyfikatorów (`getElementById()`). Często jednak nie mamy takiej możliwości (np. dostęp do wierszy tabeli tworzonej w pętli). W takich przypadkach mogą się przydać metody przechodzenia po drzewie dokumentu. Wstaw poniższą tabelkę nad formularzem.

```
<table>
    <tbody>
        <tr><td>Wiersz 1</td></tr>
        <tr><td>Wiersz 2</td></tr>
        <tr><td>Wiersz 3</td></tr>
        <tr><td>Wiersz 4</td></tr>
        <tr><td>Wiersz 5</td></tr>
        <tr><td>Wiersz 6</td></tr>
        <tr><td>Wiersz 7</td></tr>
    </tbody>
</table>
```

28. Aby uatrakcyjnić jej wygląd pokolorujemy co drugi wiersz. Dodaj do pliku ze skryptami poniższą funkcję, gdzie `i` to licznik wierszy, a `e` to wiersz. Uwaga! To nie jest poprawny sposób realizacji tego zadania. W tym celu wystarczyłaby jedna reguła w CSS'ie! Ma to jedynie zilustrować mechanizm przechodzenia po drzewie dokumentu.

```
function alterRows(i, e) {
    if (e) {
        if (i % 2 == 1) {
            e.setAttribute("style", "background-color: Aqua;");
        }
        e = e.nextSibling;
        while (e && e.nodeType != 1) {
            e = e.nextSibling;
        }
        alterRows(++i, e);
    }
}
```

29. Następnie wywołaj ją w odpowiednim miejscu przekazując jako pierwszy parametr wartość 1, a jako drugi – pierwszy wiersz w tabeli (skorzystaj z funkcji `getElementsByTagName`). Jeśli wywołanie funkcji nie powoduje błędu a nie widzisz efektu, upewnij się, że funkcja jest wywołana w odpowiednim miejscu. W tym przykładzie posłużyliśmy się metodą `getElementsByTagName()`, która wyszukuje wszystkie elementy o zadanej etykiecie. Metoda ta działa w kontekście elementu na którym jest wywołana, przeszukując wszystkie węzły drzewa znajdujące się w poniżej. Wykorzystujemy również pole `nextSibling`, które zwraca następny węzeł będący rodzeństwem aktualnego. Innymi metodami pozwalającymi na przechodzenie po węzłach są `firstChild`, `lastChild`, `previousSibling`, `parentNode` oraz `childNodes`. Należy również mieć na uwadze, iż w drzewie dokumentu węzły to nie tylko elementy, ale też na przykład komentarze czy tekst! Informację o typie węzła przechowuje pole `nodeType`, którego wartość 1 oznacza element.
30. W kolejnym przykładzie pokażemy w jaki sposób można manipulować zawartością strony przy użyciu języka JavaScript. Wstaw poniższe funkcje do pliku ze skryptami. Funkcje `nextNode()` oraz

`prevNode()` zwracają najbliższy element w przód lub w tył. Funkcja `swapRows()` posłuży nam aby wstawić ostatni wiersz tabeli jako pierwszy. Zauważ, że mimo iż usuwamy element z dokumentu nie jest on niszczony i można go ponownie wstawić do dokumentu.

```
function nextNode(e) {
    while (e && e.nodeType != 1) {
        e = e.nextSibling;
    }
    return e;
}

function prevNode(e) {
    while (e && e.nodeType != 1) {
        e = e.previousSibling;
    }
    return e;
}

function swapRows(b) {
    let tab = prevNode(b.previousSibling);
    let tBody = nextNode(tab.firstChild);
    let lastNode = prevNode(tBody.lastChild);
    tBody.removeChild(lastNode);
    let firstNode = nextNode(tBody.firstChild);
    tBody.insertBefore(lastNode, firstNode);
}
```

31. Wstaw do dokumentu pod tabelką a przed formularzem przycisk, który na kliknięcie wywoła metodę zamiany. Jako parametr wywołania funkcji przekaż naciśnięty przycisk. Zauważ, że metoda zadziała tylko wtedy, gdy element wywołujący zdarzenie będzie znajdował się bezpośrednio za tabelą.
32. Kolejny przykład ilustruje wykorzystanie języka JavaScript do kontroli długości wprowadzanego tekstu. W pliku *form.html* wprowadź po elemencie `textarea` obszar do wyświetlania licznika znaków. Zamień poniższy kod:

```
<tr>
    <td>Uwagi</td>
    <td>
        <textarea rows="5" cols="15" name="uwagi"></textarea>
    </td>
</tr>
```

na:

```
<tr>
    <td>Uwagi</td>
    <td>
        <textarea rows="5" cols="15" name="uwagi"
            onkeyup="cnt(this.form.uwagi, document.getElementById('zostalo'), 150)">
</textarea>

    </td>
</tr>
<tr>
    <td>Pozostało <span id="zostalo">150</span> znaków</td>
</tr>
```

33. Dodaj do pliku *form_check.js* kod funkcji `cnt()` i przetestuj wprowadzoną funkcjonalność.

```
function cnt(form, msg, maxSize) {
    if (form.value.length > maxSize)
        form.value = form.value.substring(0, maxSize);
    else
        msg.innerHTML = maxSize - form.value.length;
}
```