

✓ Wykonano

Dla ciekawskich

Jak działa asynchroniczność w JavaScript



1. Tworzenie animacji

1. Utwórz dokument HTML o nazwie `index.html` i następującej zawartości:

```

1.  <!-- @author Stanisław Polak <polak@agh.edu.pl> -->
2.
3.  <!DOCTYPE html>
4.  <html lang="en">
5.
6.  <head>
7.      <meta charset="UTF-8">
8.      <meta name="viewport" content="width=device-width, initial-scale=1.0">
9.      <title>Animation</title>
10. </head>
11.
12. <body>
13.     <form onsubmit="event.preventDefault();">
14.         <h2>requestAnimationFrame()</h2>
15.         <label for="counter">Counter</label>
16.         <output id="counter" style="font-size: 4vh; color: red;">0</output>
17.         <br>
18.         <button id="start" onclick="startAnimation()">Start</button>
19.         <button id="stop" disabled onclick="stopAnimation()">Stop</button>
20.         <!-- ***** -->
21.         <hr>
22.         <h2>Time-consuming calculations in the main thread</h2>
23.         <label for="result_main">Result:</label>
24.         <output id="result_main">0</output>
25.         <br>
26.         <label for="iterations_main">Number of iterations:</label>
27.         <input id="iterations_main" type="text" value="50"
onfocus="document.forms[0].result_main.value ='0'">
28.         <button
29.             onclick="document.forms[0].result_main.value =
calculatePrimes(document.forms[0].iterations_main.value || 50)">Run
30.             calculations</button>
31.         <!-- ***** -->
32.         <h2>Time-consuming calculations in a separate thread</h2>
33.         <label for="result_worker">Result:</label>
34.         <output id="result_worker">0</output>
35.         <br>
36.         <label for="iterations_worker">Number of iterations:</label>
37.         <input id="iterations_worker" type="text" value="50"
onfocus="document.forms[0].result_worker.value ='0'">
38.         <button
39.             onclick="calculatePrimesInBackground(document.forms[0].iterations_worker.value ||
50)">Run
40.             calculations</button>
41.     </form>
42.     <script>
43.         var animation;
44.         var counter = 0;
45.
46.         // Source:
https://udn.realityripple.com/docs/Tools/Performance/Scenarios/Intensive_JavaScript
47.         function calculatePrimes(iterations) {
48.             var primes = [];
49.             for (var i = 0; i < iterations; i++) {
50.                 var candidate = i * (1000000000 * Math.random());
51.                 var isPrime = true;
52.                 for (var c = 2; c <= Math.sqrt(candidate); ++c) {
53.                     if (candidate % c === 0) {
54.                         // not prime
55.                         isPrime = false;
56.                         break;

```

```

57.         }
58.     }
59.     if (isPrime) {
60.         primes.push(candidate);
61.     }
62. }
63. return primes;
64. }
65.
66. function calculatePrimesInBackground(iterations) {
67.     window.alert('Implement missing functionality')
68. }
69.
70. function startAnimation() {
71.     document.forms[0].start.disabled = true;
72.     document.forms[0].stop.disabled = false;
73.     animation = window.requestAnimationFrame(step);
74. }
75.
76. function step() {
77.     document.forms[0].counter.value = counter++;
78.     animation = window.requestAnimationFrame(step);
79. }
80.
81. function stopAnimation() {
82.     document.forms[0].start.disabled = false;
83.     document.forms[0].stop.disabled = true;
84.     window.cancelAnimationFrame(animation)
85. }
86. </script>
87. </body>
88.
89. </html>

```

2. Wyświetl go w przeglądarce, a następnie:

1. Uruchom licznik — naciśnij przycisk *Start*
 2. **W sekcji "Time-consuming calculations in the main thread"**, uruchom obliczenia przyciskiem *Run calculations* i zaobserwuj, czy obliczenia mają negatywny wpływ na szybkość działania licznika?
 3. Sprawdź, czy uruchomienie obliczeń dla znacznie większej liczby iteracji — w polu *iterations_main* ("Number of iterations") wpisz np. 2000 zamiast 50 — ma jakikolwiek wpływ na szybkość działania licznika?
 4. Zatrzymaj licznik (przycisk *Stop*), następnie ponownie uruchom obliczenia i sprawdź, czy GUI reaguje na akcje użytkownika, tzn. czy, w trakcie trwania obliczeń, przycisk *Start*, natychmiastowo, wykrywa fakt jego naciśnięcia i uruchamia licznik?
3. Zawarty w dokumencie skrypt używa metody [window.requestAnimationFrame\(callback\)](#) do cyklicznej modyfikacji wartości licznika. Przeczytaj opis timerów [window.setInterval\(callback, delay\)](#) oraz [window.setTimeout\(callback, delay\)](#) — użyjesz ich w zadaniu 3; ewentualnie obejrzyj film

JavaScript setTimeout & setInterval In 90 Seconds #JavaScriptJanuary



4. Wywnioskuj: czy w przypadku tych wszystkich trzech metod, treść funkcji zwrotnych (ang. callbacks) wykonuje się w osobnym wątku, czy w wątku głównym? Zweryfikuj swoją hipotezę — przeczytaj artykuły:
1. [Intensive JavaScript](#)
 2. [How JavaScript Timers Work](#)
5. Przeczytaj opis [wątków roboczych](#) lub obejrzyj film

JavaScript Web Workers Explained



6. Utwórz plik `worker.js` — wątek roboczy, w procedurze obsługi zdarzenia `'message'`, ma:
1. Wykonać czasochłonne obliczenia — skopiuj treść funkcji `calculatePrimes(iterations)` — dla odebranej liczby iteracji
 2. Odesłać wynik obliczeń (tablica *primes*) do wątku głównego
 3. Zamknąć (bieżący) wątek
7. Zaimplementuj brakującą funkcjonalność w `calculatePrimesInBackground(iterations)`:
1. Utworzenie wątku roboczego
 2. Rejestracja procedury obsługi zdarzenia `'error'`. Procedura, korzystając z `window.alert()`, wyświetla szczegóły błędu
 3. Rejestracja procedury obsługi zdarzenia `'message'`. Procedura, w oparciu o odebrane dane, modyfikuje zawartość pola `'result_worker'` formularza HTML
 4. Wysłanie wątkowi roboczemu wartości parametru `iterations`
8. Sprawdź, czy znaczące zwiększenie liczby iteracji — wpisz wartość 2000 do pola `iterations_worker`, a następnie uruchom obliczenia — ma nadal negatywny wpływ na szybkość działania licznika, bądź czy GUI, z opóźnieniem, reaguje na akcje użytkownika?

Jeżeli konsola przeglądarki wyświetla napis

```
Uncaught DOMException: Failed to construct 'Worker': Script at 'file:///path/to/worker.js' cannot be accessed from origin 'null'.
    at calculatePrimesInBackground (file: ...)
    at HTMLButtonElement.onclick (file: ...)
    ...
```

to uruchom serwer WWW za pomocą komendy `python3 -m http.server`, a następnie wpisz w przeglądarce adres <http://localhost:8000/>



2. DOM Living Standard

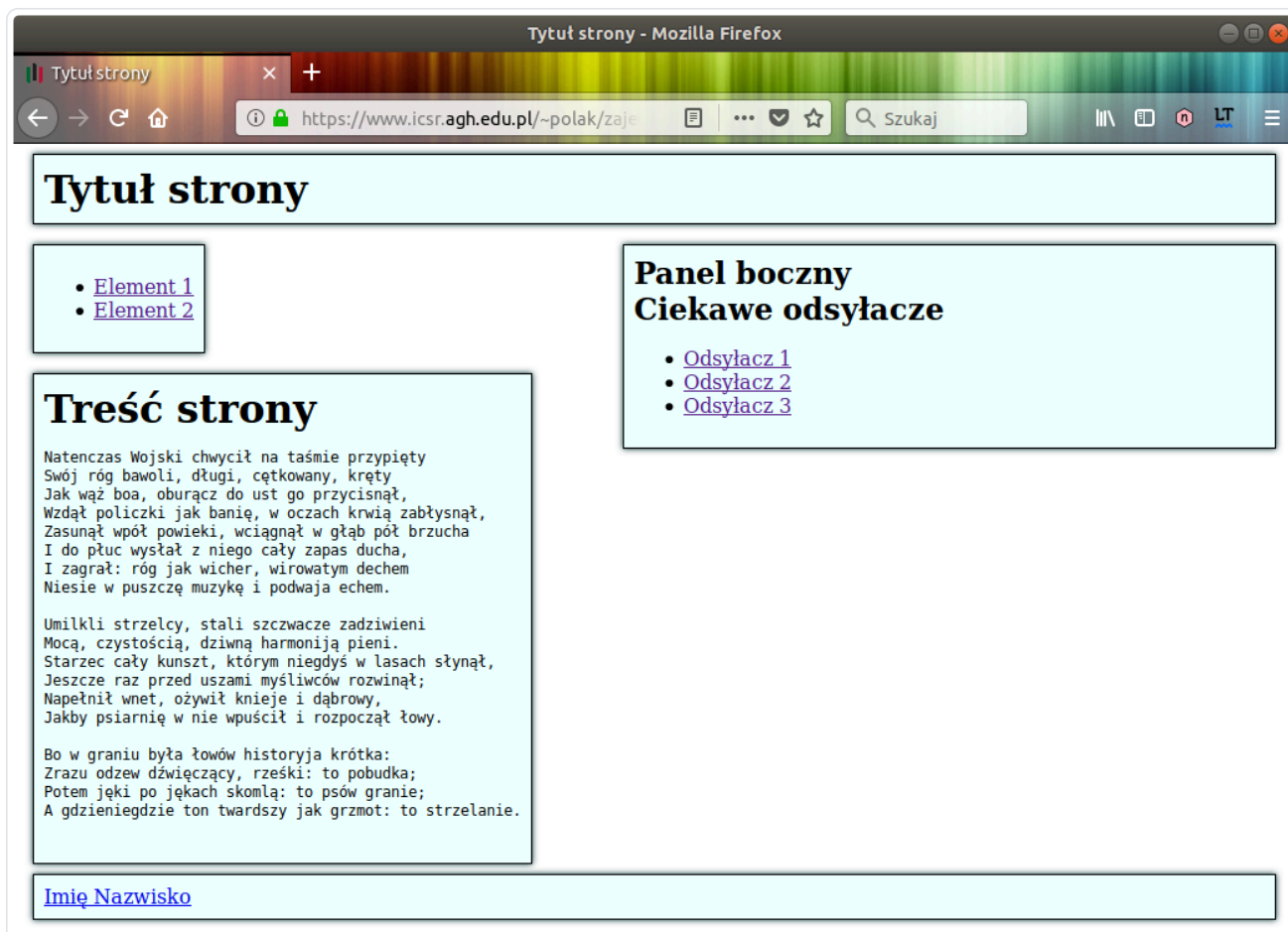
W powyższym zadaniu, jak i na poprzednich ćwiczeniach, używany był DOM poziomu 0 (DOM 0) — został on opracowany przez nieistniejącą już firmę Netscape w tym samym czasie, w którym opracowano JavaScript. Oferuje on dostęp do kilku elementów HTML, przede wszystkim formularzy oraz obrazów.

Obecnie jest używany standard DOM autorstwa organizacji **W3C**. Przez wiele lat był rozwijany jako **zbiór poziomów**, a teraz, jest on utrzymywany jako **DOM Living Standard** — do wykonania poniższego zadania użyjemy właśnie tego standardu.

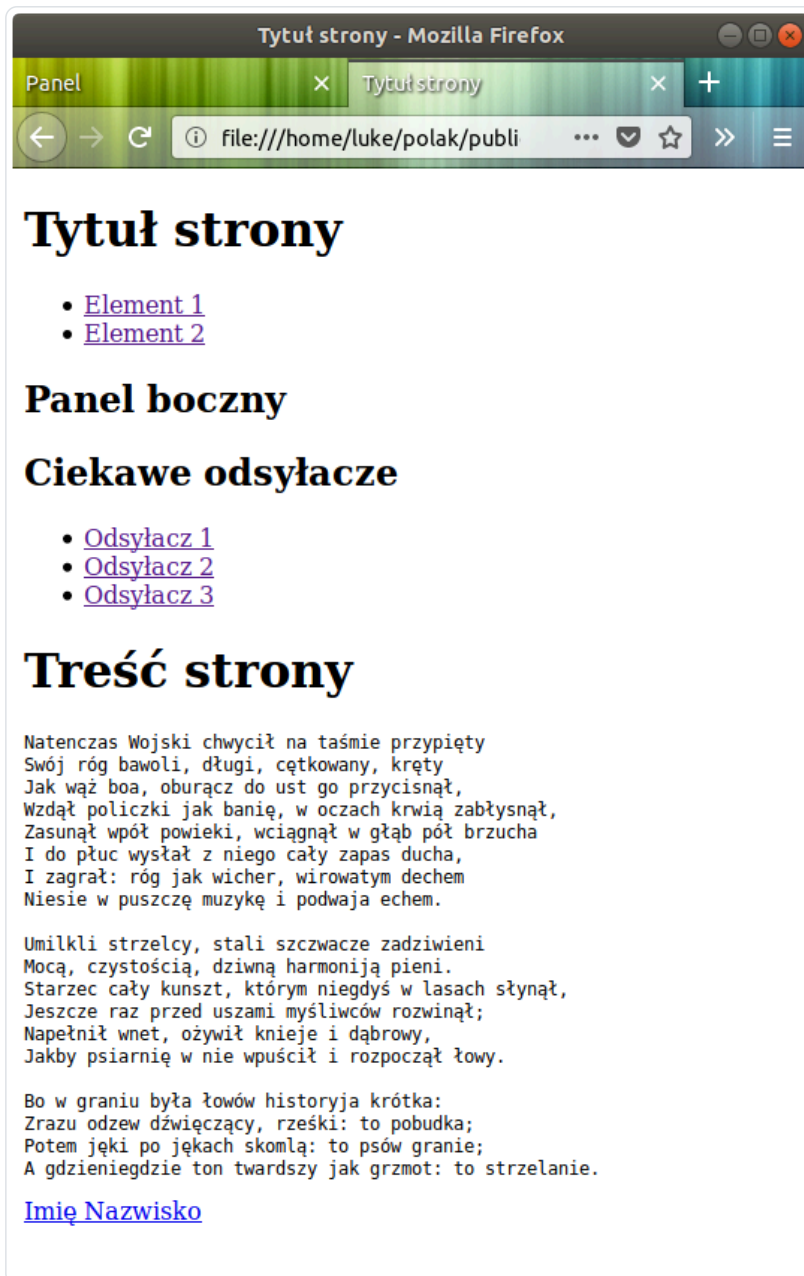
2.1. Modyfikowanie wyglądu strony WWW

- Dostęp do zawartości dokumentu HTML należy uzyskać za pomocą metod **DOM Living Standard**:
 - `document.getElementById()`
 - `document.querySelector()`
 - `document.getElementsByTagName()`
 - `document.querySelectorAll()`
- Stylowanie należy zrealizować korzystając z własności **'element.style.własność'** lub **'element.classList'** — sterowania stylowaniem **nie można** zaimplementować poprzez podpięcie lub odpięcie arkusza 'sheet.css'
- Obsługę zdarzeń proszę zrealizować w oparciu o metodę `addEventListener()`.

1. Zmodyfikuj pierwszy przykładowy dokument HTML, który był zamieszczony w jednym z poprzednich konspektów zajęć — rozszerz go o formularz zawierający trzy przyciski: "Set", "Delete" oraz "Add"
2. Stwórz JavaScriptową wersję arkusza 'sheet.css' utworzonego na pierwszych ćwiczeniach — po naciśnięciu przycisku "Set" skrypt styluje elementy dokumentu HTML; w rezultacie powinniśmy więc otrzymać stronę o następującym wyglądzie:



3. W przypadku naciśnięcia przycisku "Delete", skrypt przywraca oryginalny (początkowy) wygląd strony, tzn. usuwa wszystkie style CSS związane z elementami (usuwa stylowanie elementów) — w tym przypadku strona powinna wyglądać jak na poniższym zrzucie



ekranowym:

2.2. Modyfikowanie zawartości strony WWW

Proszę nie używać `innerHTML` — patrz [kwestie bezpieczeństwa i wydajności](#) — należy korzystać z metod: `appendChild()`, `createElement()`, `createTextNode()` (lub własność `textContent`) itp.

- Treść koncertu Wojskiego na rogu ma być przechowywana w skrypcie
- Zawartość elementu `main` jest, początkowo, pusta
- Po kolejnych naciśnięciach klawisza "Add" pojawiają się kolejne akapity
- Jeżeli nie ma już nowych akapitów do wyświetlenia, to przycisk "Add" powinien zostać [dezaktywowany](#).



3. React

... zatrzymajmy się na chwilę przy pojęciu samych komponentów. Ja przez to określenie rozumiem pojedynczy element, który posiada swoją reprezentację na stronie internetowej (widok) oraz posiadający pewną logikę, która nim steruje.

Po co nam są komponenty podczas tworzenia aplikacji? Po pierwsze pomagają nam utrzymywać czysty kod, w którym nie mamy zbędnych powtórzeń — można o tym myśleć jak o odpowiedniku enkapsulacji z programowania obiektowego. Po drugie jak będziemy konsekwentni, to tworzenie aplikacji sprowadzi się do wybierania odpowiednich klocków (komponentów) i łączenia ich między sobą. Po trzecie i ostatnie jeśli będziemy musieli wykonać zmianę, to zrobimy ją w jednym miejscu.

— Cytat ze strony **Czym są Web Components?**

Komponenty React

React JS - kurs w 60 minut



Komponenty Webowe

Web Components co to jest i czy dalej potrzebujesz framework'a?



1. Zaznajom się z poniższymi przykładami — niektóre z nich będą przydatne 😊

Przykład 1

```

1. <!-- @author Stanisław Polak <polak@agh.edu.pl> -->
2.
3. <!DOCTYPE html>
4. <html>
5.   <head>
6.
7.     <script src="https://cdn.jsdelivr.net/npm/react@latest/umd/react.development.js"
8.       crossorigin=""></script>
9.     <script src="https://cdn.jsdelivr.net/npm/react-dom@latest/umd/react-
10.       dom.development.js"
11.       crossorigin=""></script>
12.
13.     <!-- Don't use this in production: -->
14.     <script src="https://cdn.jsdelivr.net/npm/@babel/standalone/babel.min.js">
15. </script>
16.     <!-- See https://babeljs.io/docs/babel-standalone -->
17.     <title>
18.       Example 1
19.     </title>
20.   </head>
21.   <body>
22.     <div id="root">
23.       <!--
24.       Kontener dla komponentu.
25.       React renderuje wyspecyfikowany kod HTML wewnątrz tzw. kontenera,
26.       tj. wybranego przez nas elementu strony internetowej.
27.       -->
28.     </div>
29.     <!-- Komponent 'Hello' -->
30.     <script type="text/babel">
31.       // Logika komponentu
32.
33.       // Komponent klasowy
34.       class Hello extends React.Component {
35.         render() {
36.           return <h1>Hello World!</h1>;
37.         }
38.       }
39.
40.       //Komponent funkcyjny
41.       /*
42.       function Hello() {
43.         return <h1>Hello World!</h1>;
44.       }
45.       */
46.
47.       const container = document.getElementById('root'); // Pobieranie referencji
48.       na kontener
49.       const root = ReactDOM.createRoot(container); // Tworzenie korzenia
50.       React-a dla podanego kontenera
51.       root.render(<Hello />); // Renderowanie komponentu
52.     </script>
53.     <!--
54.     Note: this page is a great way to try React but it's not suitable for
55.     production.
56.     It slowly compiles JSX with Babel in the browser and uses a large development
57.     build of React.
58.
59.     Read this page for starting a new React project with JSX:
60.     https://react.dev/learn/start-a-new-react-project

```

```
56.         Read this page for adding React with JSX to an existing project:
57.         https://react.dev/learn/add-react-to-an-existing-project
58.         -->
59.         </body>
60.         </html>
```

Przykład 2

```

1. <!-- @author Stanisław Polak <polak@agh.edu.pl> -->
2.
3. <!DOCTYPE html>
4. <html>
5.   <head>
6.
7.     <script src="https://cdn.jsdelivr.net/npm/react@latest/umd/react.development.js"
8.       crossorigin=""></script>
9.     <script src="https://cdn.jsdelivr.net/npm/react-dom@latest/umd/react-
10.       dom.development.js"
11.       crossorigin=""></script>
12.
13.     <!-- Don't use this in production: -->
14.     <script src="https://cdn.jsdelivr.net/npm/@babel/standalone/babel.min.js">
15.   </script>
16.   <title>
17.     Example 2
18.   </title>
19. </head>
20. <body>
21.   <div id="root">
22.     <!--
23.       Kontener dla komponentu.
24.       React renderuje wyspecyfikowany kod HTML wewnątrz tzw. kontenera,
25.       tj. wybranego przez nas elementu strony internetowej.
26.     -->
27.   </div>
28.   <!-- Komponent 'Hello' -->
29.   <script type="text/babel">
30.     // Logika komponentu
31.
32.     // Komponent klasowy
33.     class Hello extends React.Component {
34.       render() {
35.         /* Wyświetlenie danych przekazanych za pomocą własności 'welcome' */
36.
37.         return <h1>{this.props.welcome}</h1>;
38.       }
39.     }
40.
41.     // Komponent funkcyjny
42.     /*
43.     function Hello({ welcome }) {
44.       return <h1>{welcome}</h1>;
45.     }
46.     */
47.
48.     const container = document.getElementById('root'); // Pobieranie referencji
49.     na kontener
50.     const root = ReactDOM.createRoot(container); // Tworzenie korzenia
51.     React-a dla podanego kontenera
52.     root.render(<Hello welcome="Hello World" />); // Renderowanie
53.     komponentu
54.   </script>
55. </body>
56. </html>

```

```
1. <!-- @author Stanisław Polak <polak@agh.edu.pl> -->
2.
3. <!DOCTYPE html>
4. <html>
5.   <head>
6.
7.     <script src="https://cdn.jsdelivr.net/npm/react@latest/umd/react.development.js"
8.       crossorigin=""></script>
9.     <script src="https://cdn.jsdelivr.net/npm/react-dom@latest/umd/react-
10.       dom.development.js"
11.       crossorigin=""></script>
12.
13.     <!-- Don't use this in production: -->
14.     <script src="https://cdn.jsdelivr.net/npm/@babel/standalone/babel.min.js">
15.   </script>
16.   <title>
17.     Example 3
18.   </title>
19. </head>
20. <body>
21.   <div id="root">
22.     <!--
23.       Kontener dla komponentu.
24.       React renderuje wyspecyfikowany kod HTML wewnątrz tzw. kontenera,
25.       tj. wybranego przez nas elementu strony internetowej.
26.     -->
27.   </div>
28.
29.   <pre id="output"></pre>
30.
31.   <!-- Komponent 'Hello' -->
32.   <script type="text/babel">
33.     // Logika komponentu
34.
35.     // Komponent klasowy
36.     class Hello extends React.Component {
37.       print = (event) => {
38.         let container = document.querySelector('#output');
39.         container.appendChild(document.createTextNode(this.props.welcome));
40.         container.appendChild(document.createElement('br'));
41.         container.appendChild(document.createTextNode(event._reactName));
42.         container.appendChild(document.createElement('br'));
43.       }
44.       render() {
45.         return <button onClick={this.print}>Click me</button>
46.       }
47.     }
48.
49.     // Komponent funkcyjny
50.     /*
51.     function Hello({ welcome }) {
52.       const print = (event) => {
53.         let container = document.querySelector('#output');
54.         container.appendChild(document.createTextNode(welcome));
55.         container.appendChild(document.createElement('br'));
56.         container.appendChild(document.createTextNode(event._reactName));
57.         container.appendChild(document.createElement('br'));
58.       }
59.
60.       return <button onClick={print}>Click me</button>
61.     }
```

```
60.         */
61.
62.         const container = document.getElementById('root'); // Pobieranie referencji
na kontener
63.         const root = ReactDOM.createRoot(container);      // Tworzenie korzenia
React-a dla podanego kontenera
64.         root.render(<Hello welcome="Hello World" />);    // Renderowanie komponentu
65.         </script>
66.         </body>
67.     </html>
```

Przykład 4

```

1.  <!-- @author Stanisław Polak <polak@agh.edu.pl> -->
2.
3.  <!DOCTYPE html>
4.  <html>
5.    <head>
6.
7.      <script src="https://cdn.jsdelivr.net/npm/react@latest/umd/react.development.js"
8.        crossorigin=""></script>
9.      <script src="https://cdn.jsdelivr.net/npm/react-dom@latest/umd/react-
10.        dom.development.js"
11.        crossorigin=""></script>
12.
13.      <!-- Don't use this in production: -->
14.      <script src="https://cdn.jsdelivr.net/npm/@babel/standalone/babel.min.js">
15.    </script>
16.    <title>
17.      Example 4
18.    </title>
19.  </head>
20.  <body>
21.    <div id="root">
22.      <!--
23.        Kontener dla komponentu.
24.        React renderuje wyspecyfikowany kod HTML wewnątrz tzw. kontenera,
25.        tj. wybranego przez nas elementu strony internetowej.
26.      -->
27.    </div>
28.    <!--
29.      Komunikacja pomiędzy komponentami 'EchoInput' a 'EchoOutput'.
30.      Aby mogły się komunikować, to musimy utworzyć komponent rodzicielski, tu
31.      'Echo' - będzie on przechowywał dane
32.    -->
33.    <script type="text/babel">
34.      // Komponent 'EchoInput' - odpowiada za wczytanie danych -->
35.      class EchoInput extends React.Component {
36.        constructor(props) {
37.          super(props);
38.          this.handleChange = this.handleChange.bind(this);
39.        }
40.
41.        // Obserwator zdarzenia 'Change' - ta metoda jest wywoływana, gdy pojawi
42.        się zdarzenie 'Change'
43.        handleChange(e) {
44.          this.props.handleChange(e.target.value); // Wywołuję metodę
45.          'handleChange()' zdefiniowaną w komponencie rodzicielskim 'Echo', w efekcie
46.          następuje przekazanie wczytanych danych z komponentu (potomnego) 'EchoInput' do
47.          komponentu rodzicielskiego 'Echo'
48.        }
49.
50.        render() {
51.          // Gdy wprowadzę nowy znak w '<input>', to generowane jest zdarzenie
52.          'Change'
53.          // Informacje nt. zdarzeń – https://www.pluralsight.com/guides/use-
54.          plain-javascript-events-in-react
55.          return (
56.            <input value={this.props.text} onChange={this.handleChange}
57.            />
58.          );
59.        }
60.      }
61.    </script>

```

```

52.         // Komponent 'EchoOutput' - odpowiada za wypisanie wczytanych danych -->
53.         class EchoOutput extends React.Component {
54.             render() {
55.                 // Komponent 'EchoOutput' odbiera przekazaną wartość (zmienna
                    'text'), z komponentu 'Echo', za pomocą obiektu 'props'
56.                 return (
57.                     <div> Output: {this.props.text}</div>
58.                 );
59.             }
60.         }
61.
62.         // Komponent 'Echo' - wersja klasowa
63.         class Echo extends React.Component {
64.             constructor(props) {
65.                 super(props);
66.                 this.handleChange = this.handleChange.bind(this);
67.                 this.state = { text: '' }; // Zmienna stanowa 'text' - przechowuje
                    wartość wczytaną za pomocą 'EchoInput'
68.             };
69.
70.             // Metoda do modyfikacji zmiennej stanowej 'text' - będzie ona wywoływana
                    z komponentu potomnego 'EchoInput'
71.             handleChange(newText) {
72.                 this.setState({ text: newText });
73.             }
74.
75.             render() {
76.                 return (
77.                     <React.Fragment>
78.                         <h1>Komponent 'EchoInput'</h1>
79.                         <EchoInput text={this.state.text} handleChange=
                            {this.handleChange} />
80.                         <h1>Komponent 'EchoOutput'</h1>
81.                         {
82.                             // W celu przekazania wartości (zmiennej stanowej 'text')
                                z komponentu rodzicielskiego 'Echo' do komponentu potomnego 'EchoOutput'. musimy użyć
                                własności komponentu potomnego
83.                             <EchoOutput text={this.state.text} />
84.                         }
85.                     </React.Fragment>
86.                 );
87.             }
88.         }
89.
90.         // Komponent 'Echo' - wersja funkcyjna
91.         /*
92.         function Echo() {
93.             const [text, setText] = React.useState(''); // Zmienna stanowa 'text' -
                    przechowuje wartość wczytaną za pomocą 'EchoInput'
94.             const handleChange = (newText) => { // Metoda do modyfikacji
                    wartości zmiennej stanowej 'text' - będzie ona wywoływana z komponentu potomnego
                    'EchoInput'
95.                 setText(newText);
96.             }
97.
98.             return (
99.                 <React.Fragment>
100.                     <h1>Komponent 'EchoInput'</h1>
101.                     <EchoInput text={text} handleChange={handleChange} />
102.                     <h1>Komponent 'EchoOutput'</h1>
103.                     {

```



```

104. // W celu przekazania wartości (zmiennej stanowej 'text') z
    komponentu rodzicielskiego 'Echo' do komponentu potomnego 'EchoOutput'. musimy użyć
    własności komponentu potomnego
105.     }
106.     <EchoOutput text={text} />
107. </React.Fragment>
108. );
109. }
110. */
111.
112. const container = document.getElementById('root'); // Pobieranie referencji
    na kontener
113. const root = ReactDOM.createRoot(container); // Tworzenie korzenia
    React-a dla podanego kontenera
114. root.render(<Echo />); // Renderowanie komponentu
115. </script>
116. </body>
117. </html>

```

2. Zmodyfikuj stronę z zadania nr 1 — dodaj zestaw, niezależnych, liczników — każdy z nich to [komponent React](#)

Komponent "Counter"

- Widok:
 - Element blokowy, np. 'div', z zielonym tłem
 - Posiada elementy składowe:
 - Napis "Counter→"
 - Aktualna wartość licznika (w kolorze czerwonym)
 - Przycisk "Start"
 - Przycisk "Stop"
- Logika:
 - Komponent, [co delay ms](#), inkrementuje, a następnie wyświetla, wartość wewnętrznego licznika. Początkowa wartość licznika (*initial*) oraz liczba milisekund (*delay*) są parametrami komponentu — dla przykładu poniżej: `<Counter initial="10" delay="1000"/>` `<Counter initial="15" delay="500"/>`
 - Przycisk "Start" uruchamia licznik, a przycisk "Stop" zatrzymuje

Zainteresowane osoby mogą, [nieobowiązkowo / dodatkowo](#), zaimplementować licznik jako **komponent webowy**

Przykładowa strona

setInterval() / setTimeout()

Counter→ 10

Start Stop

Counter→ 15

Start Stop

requestAnimationFrame()

Counter→ **0**

Time-consuming calculations in the main thread

Result: 0

Number of iterations:

Time-consuming calculations in a separate thread

Result: 0

Number of iterations:

Dla ciekawskich

- Jeśli React Cię zainteresował i chcesz się dowiedzieć czegoś więcej na jego temat, to zaznajom się z poniższym filmem.

React - kurs podstaw w 2h krok po kroku!



- Czy warto, nadal, uczyć się Reacta?

Czy Next.js WYPRZE Reacta?



- Narzędzia do tworzenia testów w JavaScript

DLACZEGO testujemy kod? Narzędzia do tworzenia testów w JavaScript - Jest, Playwright



4. Zadanie

Napisz skrypt, który:

- Modyfikuje treść lub wygląd dokumentu HTML z poprzednich ćwiczeń używając DOM Living Standard
- Obsługuje zdarzenia za pomocą odpowiedniego [obserwatora zdarzeń](#)
- Korzysta z metod: `window.requestAnimationFrame()`, `window.setInterval()` lub `window.setTimeout()`

Szczegóły zostaną określone **na początku zajęć**

Edytuj zadanie

Usuń zadanie