AGENDA

- DOM
- window - kontrola nad oknem przeglądarki
- document - budowa dokumentu html
- wyszukiwanie i modyfikowanie elementów na stronie www
- zarządzanie zdarzeniami
- synchroniczność oraz asynchroniczność
- zarządzanie czasem - setTimeout, setInterval
- Obietnice i obserwatorzy

DOM - DOCUMENT OBJECT MODEL

- interface dla dokumentów HTML oraz XML
- pozwala na zmiany struktury, stylu i zawartości strony
- reprezentuje strone w postaci węzłów (node) oraz obiektów
- dzięki niemu jesteśmy w stanie oddziaływać na stronę językami programowania np JS

DOSTĘP DO DOM

WINDOW - OKNO PRZEGLĄDARKI

- pozwala na dostęp do localStorage i sessionStorage
- pozwala na przewinięcie strony do wybranego miejsca
- daje nam dostęp do dokumentu
- pozwala ustawić rozmiar okna, schować toolbary itp
- pozwala otworzyć bądz też zamknąć nowe okno

DOCUMENT - GŁÓWNY DOKUMENT STRONY

- za jego pomocą znajdujemy elementy strony i odziałujemy na nie
- dzięki niemu kontrolujemy style strony
- daje nam dostęp do URL, domeny czy ciasteczek
- daje nam dostęp do zdarzeń
- pozwala tworzyć nowe elementy

PRZECHOWYWANIE DANYCH - LOCALSTORAGE I SESSIONSTORAGE

- setItem
- getItem
- removeItem
- clear

CWICZENIE 1

WYSZUKIWANIE ELEMENTÓW NA STRONIE

- id - `document.getElementById()`
- tag - `document.getElementsByTagName()`
- class - `document.getElementsByClassName()`
- css selector - `document.querySelector()`
- css selector all - `document.querySelectorAll()`

DZIAŁANIA NA ELEMENTACH

- dodawanie i usuwanie atrybutów
- dodawanie i usuwanie innych elementów
- dodanie nasłuchu zdarzeń

ĆWICZENIE 2

EVENTS - ŹRÓDŁA ZDARZEŃ

- akcja użytkownika
- wygenerowane przez asynchroniczne zadanie
- wygenerowanie za pomocą bezposredniej instrukcji w kodzie

SPOSOBY PRZECHWYTYWANIA ZDARZEŃ

- wstrzykiwany w html event handler
- własciwości eventHandlera
- funkcja addEventListener

EVENT BUBBLING

- każde zdarzenie propaguje się od dziecka do rodzica
- możemy w każdej chwili zatrzymać propagacje za pomocą funkcji `stopPropagation()`

ĆWICZENIE 3

- Przygotuj formularz który dodaje przepisy do listy przepisów
- Formularz powinien miec możliwośc dynamicznego dodawania składników oraz kroków
- Spraw by przepisy zapisywały sie w localStorage

SYNCHRONICZNOŚĆ I ASYNCHRONICZNOŚĆ

setTimeout

```
console.log(1)
setTimeout(function(){ console.log(2); }, 1000);
console.log(3)
```

setInterval

```
const IntervId = setInterval(function(){ console.log('x')}, 1000);
clearInterval(nIntervId);
```

OBIETNICE - PROMISES

```
let myFirstPromise = new Promise((resolve, reject) => {
  setTimeout( function() {
    resolve("Success!")
  }, 250)
})

myFirstPromise.then((successMessage) => {
    console.log(successMessage)
});
```

OBIETNICE MAJA 3 STANY

1. pending
2. fulfilled
3. rejected

MOŻEMY TWORZYĆ ŁAŃCUCHY OBIETNIC

```
myFirstPromise.then((successMessage) => {
    return 'jupi! '+ successMessage
}).then((modMessage) => {
    console.log(modMessage)
});
```

ĆWICZENIE 4

OBSERWATORY

INSTALACJA

```
npm install rxjs
```

```
import { fromEvent } from 'rxjs';
fromEvent(document, 'click').subscribe(() =>
console.log('Clicked!'));
```

OBSERVABLE

```
const observable = new Observable(subscriber => {
  subscriber.next(1);
  setTimeout(() => {
    subscriber.next(4);
    subscriber.complete();
  }, 1000);
});
```

OBSERVER

```
const observer = {
  next: x => console.log('Observer got a next value: ' + x),
  error: err => console.error('Observer got an error: ' + err),
  complete: () => console.log('Observer got a complete
notification'),
};
observable.subscribe(observer);
```

OPERATOR

```
observable.pipe(
map(x => x*x),
filter(x => x>2)
).subscribe(observer);
```

SUBSCRIPTION

```
const sub = observable.pipe(map(x => x*x)).subscribe(observer);
sub.unsubscribe();
```

SUBJECT

```
subject.subscribe({
  next: (v) => console.log(`observerA: ${v}`)
});
subject.subscribe({
  next: (v) => console.log(`observerB: ${v}`)
});
subject.next(1);
subject.next(2);
```

ĆWICZENIE 5