9. Struktury danych w JavaScript

Wyzwania:

- Dowiesz się czym są tablice, obiekty i pętle.
- Nauczysz się wykonywać te same operacje na wielu elementach DOM.
- Rozwiniesz swoją grę w papier, kamień i nożyce!

9.3. PROJEKT: Rozbudowa gry!

9.1. Tablice, obiekty i pętle

W poprzednim module poznaliśmy sporą część podstaw JavaScriptu — zmienne i operacje na nich, strukturę if/else, funkcje oraz wyszukiwanie i zmiany elementów DOM. W tym tygodniu poznamy jeszcze tablice, obiekty i pętle, które znacznie poszerzą nasze możliwości w pisaniu skryptów.

Tablice, czyli arrays

Korzystaliśmy już ze zmiennych, ale do tej pory każda zmienna miała tylko jedną wartość. Zmienna może jednak zawierać również tablicę, którą można wytłumaczyć jako zbiór wartości.

Metafora — tablica

Jak zapewne pamiętasz, w poprzednim module porównywaliśmy zmienną do pudełka z etykietą.

Tablicę możemy sobie w takim razie wytumaczyć, jako bardzo długie pudełko, w którym znajdują się kolejno ułożone mniejsze pudełka.

Możemy łatwo sprawdzić, ile mamy małych pudełek, ale nie mają one etykiet.

Jak więc zatem wskazać pudełko, które nas interesuje? Musimy tylko wiedzieć, na którym miejscu w szeregu znajduje się to pudełko. Pozycję danego pudełka (pierwsze, drugie, czwarte...) nazywamy indeksem.

Otarliśmy się już o to zagadnienie, kiedy korzystaliśmy z `document.getElementsByClassName` oraz `element.children`. W obu przypadkach rezultatem tych funkcji nie był pojedynczy element DOM, ale właśnie tablica. Dlatego używaliśmy `[0]` do wybrania pierwszego elementu.

HTML SCSS JS Result

```
'use strict';
(function(){
  /* Zaczynamy od 'use strict' oraz enkapsulacji w samowywołującą anonimową funkcję - jeśli nie pamiętasz tych pojęć, wróć do rozdziału "Zakres zmiennych" z poprzedniego modułu.

  Zadeklarujemy teraz nową zmienną, której wartością będzie tablica.
  */

  var exampleOne = [777, 'abc', 123];

  /* Aby wybrać pierwszą wartość
```

```
Let's see how the script works...

First element in array exampleOne: 777
Second element in array exampleOne: abc
Third element in array exampleOne: 123
Fourth element in array exampleOne: undefined
Length of array exampleOne: 3
Fourth element in array exampleOne: Lorem ipsum
Length of array exampleOne: 4
First element in array exampleTwo: qwerty
Length of array exampleTwo: 1
Element 2 in array exampleOne: 123
```

Resources 1× 0.5× 0.25× Rerun

Tablice posiadają wiele innych metod, poza metodą `push`, którą poznaliśmy przed chwilą. Metody te pozwalają m.in.:

- dodawać nowy element na początku tablicy,
- usuwać element z początku lub końca tablicy,
- pobierać lub usuwać dowolną grupę elementów z tablicy,
- filtrować tablicę wedle dowolnych kryteriów,
- wykonywać dowolne operacje na każdym z elementów tablicy.

Wszystkie te metody z łatwością znajdziesz w internecie za pomocą wyszukiwarki Google lub w dokumentacji MDN.

Ćwiczenie — filtrowanie tablicy

Może Cię dziwić, że nie tłumaczymy wszystkich, albo przynajmniej najbardziej popularnych metod, z których można korzystać na tablicach. Zależy nam jednak, aby jak najszybciej przejść do praktycznego korzystania z nich, zamiast tłumaczyć metody, których nie będziemy na razie wykorzystywać.

Drugim powodem jest fakt, że korzystanie z dokumentacji i przykładów znalezionych w internecie jest podstawową i kluczową umiejętnością, którą musi posiadać każdy developer. Nie musisz się uczyć na pamięć wszystkich metod i sposobów ich wykorzystania — lepiej wiedzieć, jak szybko znaleźć potrzebne metody.

W ramach ćwiczenia stwórz tablicę, która zawiera liczby i teksty. Twoim zadaniem jest przefiltrowanie tablicy w taki sposób, aby pozostały w niej tylko liczby. W tym celu:

- w dokumentacji MDN znajdź metodę `filter` dla tablic (array),
- w Google wyszukaj sposób na sprawdzenie czy wartość jest liczbą, wpisując np. "JavaScript check if value is a number".

Właśnie w ten sposób pracują i uczą się developerzy. Nie bój się korzystać z zasobów internetu. Pamiętaj tylko, że:

- dokumentacja MDN może zawierać metody czy przykłady, które działają tylko w niektórych przeglądarkach — dlatego np. na stronie metody `filter` znajdziesz rozdział "Kompatybilność z przeglądarkami" ("Browser compatibility"), w którym znajduje się informacja, które przeglądarki i od której wersji pozwalają na korzystanie z tej metody,
- przykłady znalezione w internecie, np. na StackOverflow, mogą być przestarzałe lub niepoprawne, więc musisz samodzielnie ocenić i/lub przetestować, czy dany przykład działa poprawnie. Warto też czytać dokładnie treść komentarzy, ponieważ może ona tłumaczyć np. dlaczego podany przykład kodu jest niepoprawny.

Obiekty, czyli objects

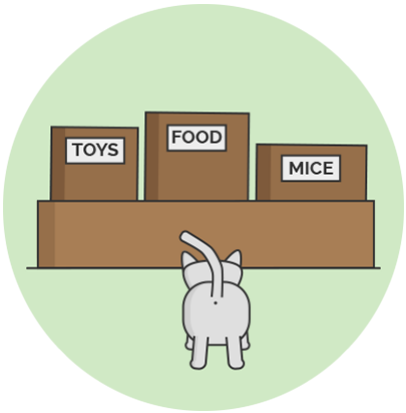
Zacznijmy od wyjaśnienia, że samo słowo "obiekt" w JS jest nieco mylące, i występuje w trzech znaczeniach:

- "zwykłe" obiekty, czyli **plain objects**, o których będziemy mówić w tym rozdziale,
- obiekty jako typ danych, sprawdzany za pomocą `typeof` — z dość skomplikowanych powodów nie tylko plain object będzie miał typ `object`, ale również np. tablica czy `null`,
- słowo "obiekt" wykorzystuje się również w zagadnieniu programowania obiektowego (OOP — Object Oriented Programming).

To jest skomplikowane i może wprowadzać w błąd, ale na razie nie będziemy się tym przejmować. W tym module mówiąc o obiektach zawsze będziemy mieli na myśli "zwykłe" obiekty. Zmieni się to dopiero w module, w którym poznamy OOP.

A więc czym są obiekty? Podobnie jak tablice, są one zestawem wartości. Różnica pomiędzy tablicami a obiektami jest taka, że o ile w tablicy mamy indeks elementu, t obiekcie mamy jego klucz, który jest nazwą czy etykietą danego elementu.

Metafora — obiekt



Podobnie jak tablica, obiekt porównamy do pudełka, w którym jest wiele małych pudełek.

W przeciwieństwie do tablicy, w obiekcie te małe pudełka nie są ułożone w jakiejś kolejności.

Zamiast tego, każde pudełko ma swoją etykietę.

Aby wybrać któreś małe pudełko, zamiast mówić "biorę pierwsze pudełko", powiemy "biorę pudełko z etykietą: zabawki dla kota".

Zobaczmy w praktyce, jak posługiwać się obiektami:

HTML SCSS JS Result

```
'use strict';
(function(){
  // Tablicę definiowaliśmy za pomocą nawiasów kwadratowych - za to obiekt definiujemy za pomocą nawiasów klamrowych:

  var exampleOne = {lorem: 'ipsum', abc: 123, foo: 'bar'};

  // Wartości pobieramy identycznie jak w tablicach, czyli podając klucz w nawiasach kwadratowych. Ważne - klamrowych nawiasów używamy tylko do
```

```
Let's see how the script works...

Element with key "abc" from object exampleOne: 123
Element with key "lorem" from object exampleOne: ipsum
Element with key "qwe" from object exampleOne: 777
Element with key "qwe" from object exampleOne: 777
```

Resources 1× 0.5× 0.25× Rerun

Jak widzisz, obiekt niewiele różni się od tablicy. Jenak zamiast kolekcji wartości, bardziej przypomina kolekcję zmiennych, ponieważ każdy z elementów ma klucz, czyli swoją "nazwę".

Obiekty nie posiadają zbyt wielu przydatnych metod. Będziemy z nich za to często korzystać do organizowana danych w "grupy", dzięki czemu będzie nam łatwiej się do nich odnosić.

Pętle, czyli loops

Pętla to nic innego, jak wykonywanie tych samych czynności w kółko określoną ilość razy, albo dla określonego zestawu elementów.

Metafora — pętla

Załóżmy, że Twój kot często chowa się w pudełku, kiedy chcesz go zabrać do weterynarza. Potrzebujesz w takim razie znaleźć kota, a masz przed sobą wiele pudełek.

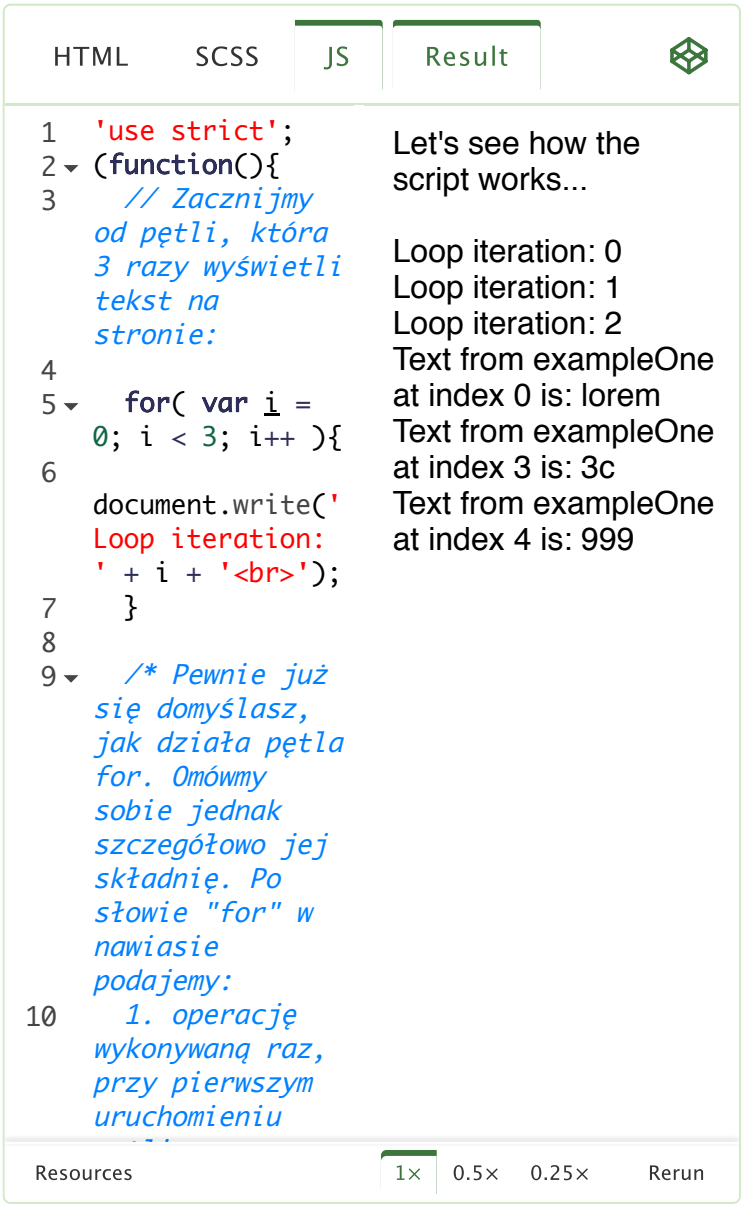
Dla każdego pudełka musisz wykonać następujący algorytm:

- jeśli pudełko jest za małe dla kota, przejść do kolejnego pudełka,
- jeśli pudełko jest oklejone taśmą klejącą i kot nie mógł do niego wejść, przejść do kolejnego pudełka,
- otworzyć pudełko, i jeśli nie ma w nim kota, przejść do kolejnego pudełka,
- jeśli jest w nim kot, wyjąć kota i nie sprawdzać kolejnych pudełek.

Nie bez powodu mówimy o pętlach właśnie teraz. Często będziemy używać pętli, aby wykonać jakąś operację dla każdego elementu tablicy lub obiektu.

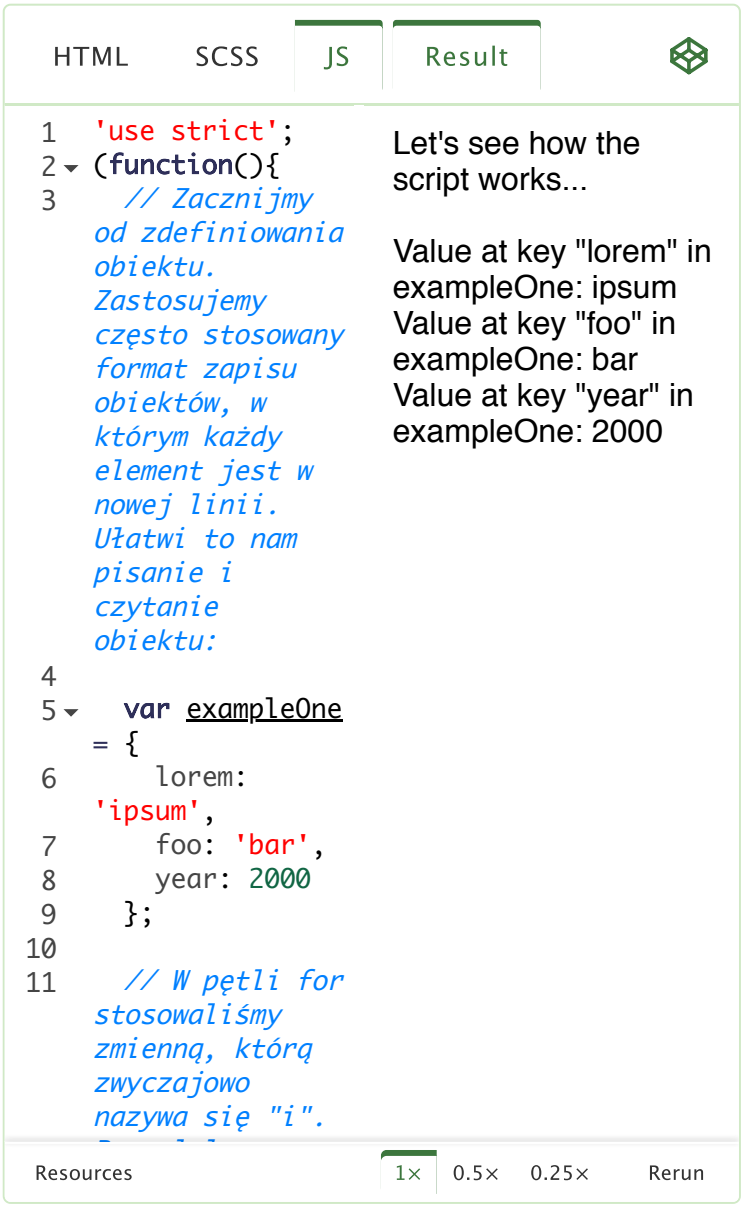
Pętla "for"

Zaczniemy od pętli "for", którą możemy wykorzystywać do wykonania fragmentu kodu wiele razy. Jest bardzo przydatna, ponieważ pozwala nie tylko wykonać jakieś operacje np. 5 razy, ale też "tyle razy, ile jest elementów w tablicy".



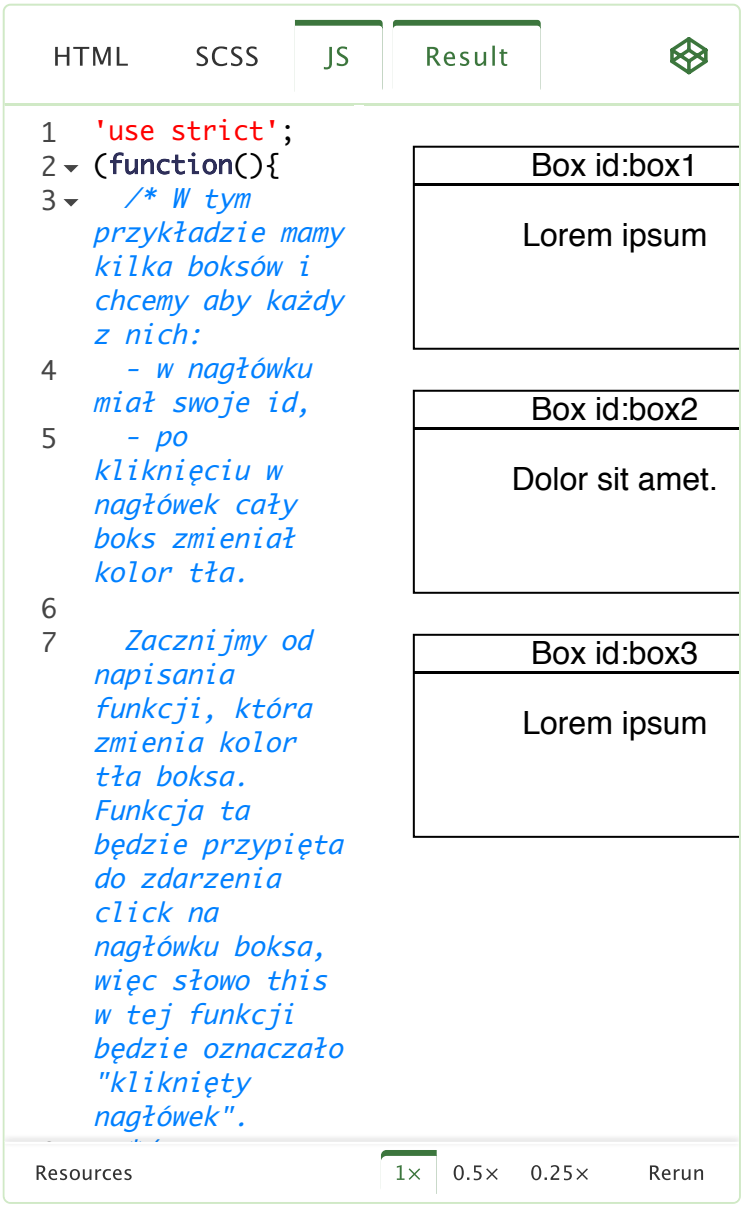
Pętla "for...in"

Drugą odmianą pętli "for" jest pętla "for...in". Dzięki niej będziemy mogli wykonać pewne operacje dla każdego elementu w obiekcie.



Pętla "for" a elementy DOM

Jak już pewnie się domyślasz, pętla będzie dla nas też bardzo przydatna, kiedy będziemy chcieli wykonać jakąś operację dla wielu elementów. Co ważniejsze, dzięki temu nasz skrypt nie musi też wiedzieć, ile będzie tych elementów — zadziała również dla elementów, które w przyszłości dodamy do kodu HTML.



Zadanie: Struktury danych

Do tej pory w tablicach i obiektach przechowywaliśmy tylko wartości liczbowe i tekstowe, jednak — tak samo jak w zmiennych — można w nich przechowywać także tablice i obiekty! W tym zadaniu zapoznamy się z wielopoziomową strukturą danych.

W poniższym fragmencie kodu znajdziesz zmienną `data`. W tej zmiennej zapisaliśmy tablicę, której elementami są obiekty. Każdy z tych obiektów ma taką samą strukturę.

```
var data = [
    {
        id: 'box1',
        title: 'First box',
        content: '<p>Lorem ipsum dolor sit amet!</p>',
        categories: ['highlighted', 'special-header', 'important']
    },
    {
        id: 'box2',
        title: 'Second box',
        content: '<p>Lorem ipsum dolor sit amet!</p>',
        categories: ['special-header', 'important']
    },
    {
        id: 'box3',
        title: 'Third box',
        content: '<p>Lorem ipsum dolor sit amet!</p>',
        categories: ['highlighted', 'important']
    },
    {
        id: 'box4',
        title: 'Fourth box',
        content: '<p>Lorem ipsum dolor sit amet!</p>',
        categories: ['highlighted']
    },
    {
        id: 'box5',
        title: 'Fifth box',
        content: '<p>Lorem ipsum dolor sit amet!</p>',
        categories: []
    },
];
```

Stwórz nowy projekt (pen na CodePenie lub repozytorium na GitHubie) i umieść w nim ten fragment kodu.

Twoim zadaniem jest wygenerowanie i wyświetlenie na stronie po jednym boksie dla każdego z obiektów, stosując poniższe zasady:

1. Wartość `id` ma być atrybutem `id` boksa.
2. Wartość `title` ma być zawartością headera.
3. Wartość `content` ma zostać wstawiona po headerze.
4. Każda z wartości w tablicy `categories` ma zostać dodana jako klasa boksa.

Cały boks najlepiej wygenerować jako jeden ciąg tekstowy i dopiero wtedy dodać go do `body` lub innego elementu.

Klasy dodane na boksie:

- `highlighted` — zmienia kolor tła całego boksa,
- `special-header` — zmienia kolor headera,
- `important` — jeśli nie ma tej klasy, boks ma mieć `opacity: 0.7;` , a jeśli ta klasa jest nadana, ma mieć `opacity: 1;` .

Pamiętaj, aby realizować zadanie krok po kroku. Pierwszym krokiem może być choćby wygenerowanie zwykłego diva z tytułem dla każdego obiektu.

Powodzenia!

Podgląd zadania

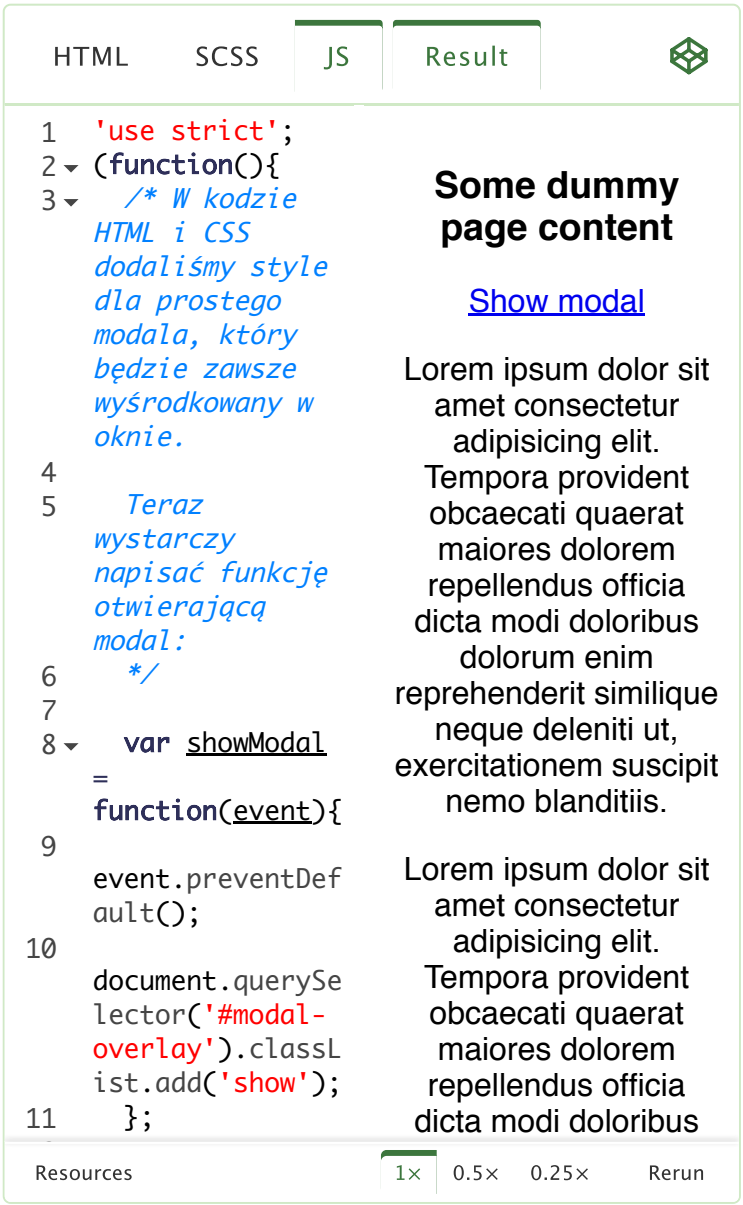
https://codepen.io/0na/

Wyślij link ✔

9.2. Modal czyli wyskakujące okienko

Modal lub popup — tak zwykle mówi się na wyskakujące okienko na stronie. Może być bardzo proste, zawierać tylko tekst i guzik zamykania. Może jednak być też bardziej złożone, np. jeśli zawiera formularz kontaktowy.

Na stronach bardzo często wykorzystuje się modale, aby przykuć uwagę użytkownika do komunikatu lub formularza. Dlatego jednocześnie przesłania się resztę strony półprzezroczystym tłem, które potocznie nazywa się overlayem.

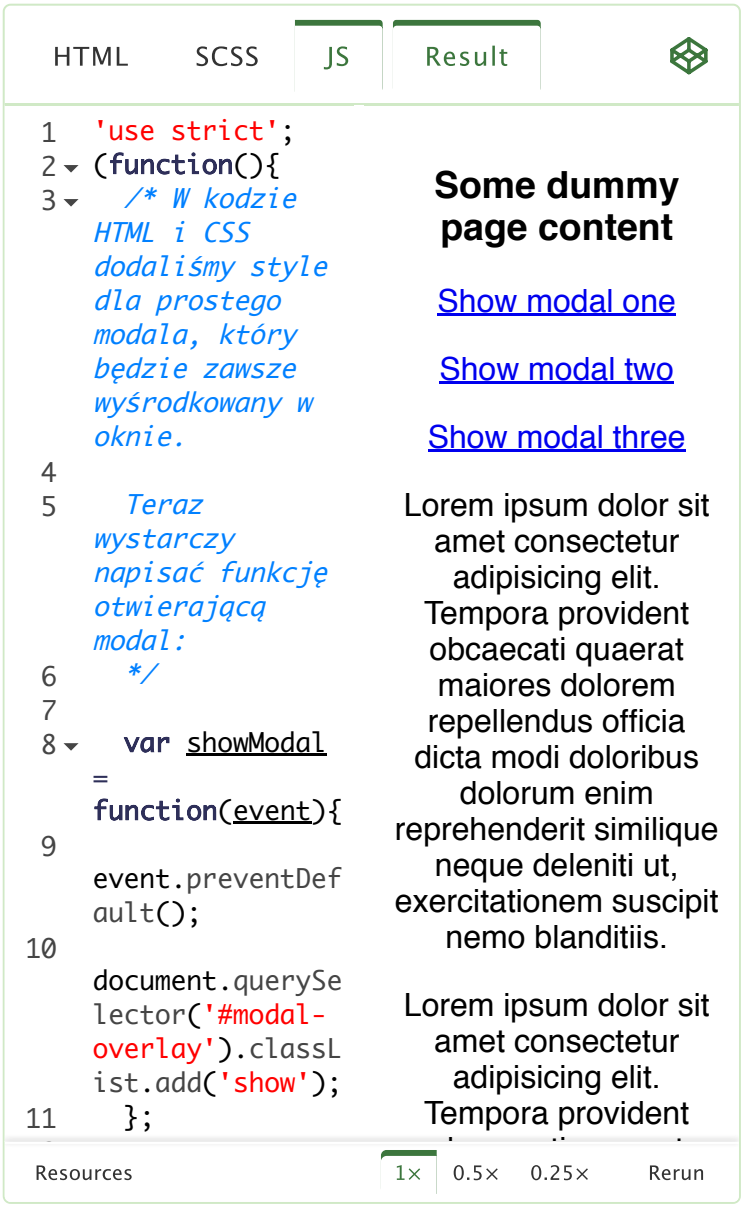


Zadanie: Obsługa wielu modali

Stwórz nowy CodePen lub projekt na GitHubie. Za chwilę skopiujesz do niego zawartość poniższego przykładu, w którym wprowadziliśmy kilka zmian:

1. Na stronie są trzy linki "Show modal", każdy z inną wartością atrybutu `href`.
2. W elemencie `.overlay` znajdują się trzy modale, każdy z innym `id`.
3. Atrybuty `href` w linkach odpowiadają atrybutom `id` modali.
4. Modale domyślnie są ukryte. Aby modal się wyświetlił, musi otrzymać klasę `show` (tak samo jak overlay).

Kod JS jest taki sam, jak w powyższym przykładzie.



Twoim zadaniem jest taka modyfikacja funkcji `showModal`, aby kliknięcie w "Show modal one":

1. Usuwało klasę `show` ze wszystkich modali,
2. Dodawało klasę `show` do modala o `id="modal-one"`,
3. Dodawało klasę `show` do overlaya (to już jest wykonywane w tej funkcji).

Oczywiście, ten sam kod ma analogicznie działać dla wszystkich linków, więc "modal-one" w powyższej instrukcji musi być pobierane z atrybutu `href` linka. Zwróć uwagę, że jego wartość zaczyna się od `#`, czyli możesz użyć całej wartości `href` jako selektora.

Pamiętaj, aby debugować swój kod krok po kroku! Powodzenia!

Podgląd zadania

https://codepen.io/0na/|

Wyślij link ✔

9.3. PROJEKT: Rozbudowa gry!

Czas na rozbudowę Twojej gry w papier-kamień-nożyce! Dodamy zarówno kilka zmian wizualnych, jak i kilka zmian w architekturze samego kodu JS. Dzięki temu Twoja gra będzie gotowa do zaprezentowania w portfolio!

To zadanie ma kilka etapów, które wykorzystują wiedzę zdobytą w tym module. Podobnie jak zadanie w poprzednim module, jest opisane dokładnie tak, aby umożliwić jego bezproblemową realizację. Wymaga jednak intensywnego korzystania ze zdobytej wcześniej wiedzy.

Nie bój się wracać do przykładów z tego i poprzedniego modułu, eksperymentować, czy rozbijać każdego etapu na małe kroki. I nie stresuj się za bardzo - dasz sobie ze wszystkim radę, tylko daj sobie czas na zrozumienie każdego z etapów.

Powodzenia!

Zadanie: Gra z modalami i przebiegiem gry

Znajdź swoje zadanie z poprzedniego modułu i otwórz je. Zaczynamy od przeniesienia naszego kodu na GitHuba.

Etap 1 — przeniesienie projektu z CodePena na GitHuba

Jeśli Twój projekt już jest na GitHubie, stwórz nowy fork tego repozytorium lub branch.

Jeżeli jednak, zgodnie z instrukcjami, do tej pory Twoje zadanie "żyło" na CodePenie, teraz przeniesiemy je na GitHuba.

1. Stwórz nowe repozytorium na GitHubie i sklonuj je na swój dysk.

2. W swoim penie z grą znajdź guzik "Export" - jest to mały guzik w dolnym prawym rogu okna. Pobierz zip ze swoim projektem i rozpakuj go do katalogu sklonowanego repozotyrium.
3. Jeśli masz katalog *scss*, zmień jego nazwę na *sass*. W przeciwnym wypadku stwórz katalog *sass*, przenieś do niego plik *css/style.css* i zmień jego rozszerzenie na *.scss*.
4. Do katalogu projektu dodaj plik *package.json* z ostatniego zadania w module 7.
5. Otwórz w terminalu katalog projektu i uruchom komendę `npm run init-project`. Ta komenda powinna zainstalować wszystkie niezbędne pakiety NPM, dodać brakujące katalogi (np. *vendor* i *js*) oraz dodać plik *.gitignore*.
6. Zapisz commit i wyślij go do zdalnego repozytorium. Przez stronę GitHuba sprawdź czy w projekcie pojawiły się pliki projektu. Pamiętaj, że w repozytorium nie powinno być katalogu *node_modules*.
7. Uruchom `npm run watch` aby móc dalej pracować ze swoim projektem.

Brawo, masz już projekt na swoim zdalnym repozytorium na GitHubie! To może być dobry moment aby przypomnieć sobie zawartość modułu 7, ponieważ od tej chwili będziesz już pracować z wykorzystaniem Gita i naszego task runnera.

Etap 2 — przywiązanie funkcji do guzików

Czas rozpocząć prace nad kodem JS. Na początek usprawnienie sposobu w jaki przywiązaliśmy funkcję `playerMove` do guzików: papier, kamień i nożyce.

1. W kodzie HTML do guzika "papier" dodaj atrybut `data-move="paper"`, i analogicznie dla "kamienia" i "nożyc" — "rock" i "scissors".
2. Dla każdego z tych guzikuj dodaj też klasę `player-move`.
3. W kodzie JS, zamiast osobnego przywiązania funkcji do każdego z guzików, stwórz pętlę przechodzącą przez wszystkie elementy z klasą `player-move`.

3.1. W pętli niech dla każdego guzika będzie przywiązana funkcja, która wywołuje funkcję `playerMove` z odpowiednim argumentem. Tym argumentem będzie wartość atrybutu `data-move`, pobranego za pomocą getAttribute.

Etap 3 — obiekt zawierający parametry gry

Dla uporządkowania kodu, stwórz zmienną `params` zawierającą obiekt. W tym obiekcie przechowuj wszystkie wartości, które obecnie są zapisane w zmiennych globalnych, np. liczbę rozgranych rund, wynik gracza i komputera, liczbę wygranych rund która powoduje wygranie gry, informację, czy gra została zakończona, itp.

Dla przykładu, jeśli do tej pory zmienna w Twoim kodzie nazywała się `roundsPlayed`, teraz będzie się nazywać `params.roundsPlayed`.

Jest to dobra praktyka — dzięki temu będziesz wiedzieć, że tylko ten obiekt może być zmieniany przez funkcje. Nie będzie trzeba już się obawiać, że w którejś funkcji nadpiszesz jedną ze zmiennych globalnych. Wprowadza to też większy porządek w kodzie i większą czytelność dla innego developera (i dla Ciebie, kiedy zajrzysz do tego kodu po dłuższym czasie).

Etap 4 — modal z wynikiem gry

Czas na dodanie do Twojej gry pierwszego modala (tak, będzie ich więcej). Do tej pory po osiągnięciu przez gracza lub komputer zadanej liczby wygranych rund, gra kończyła się wyświetleniem na stronie komunikatu, który informował kto wygrał całą rozgrywkę.

Teraz chcemy, aby ten komunikat wyświetlał się w modalu. W tym celu dodaj do strony modal, podobnie jak w zadaniu dot. modala z tego modułu. Możesz użyć tego samego kodu JS, ponieważ będzie kilka różnych modali.

Po zakończeniu gry ten sam komunikat, który wcześniej był wyświetlany na stronie, ma teraz być wstawiany do modala, a modal ma zostać pokazany.

Etap 5 — tabela przebiegu gry

W modalu z wynikiem gry chcemy - oprócz komunikatu - wyświetlać tabelę z przebiegiem gry. Będzie to prosta tabelka z kolumnami:

- numer rundy,
- ruch gracza,
- ruch komputera,
- wynik rundy,
- wynik gry po tej rundzie (np. "0-1" jeśli to pierwsza runda i wygrał komputer).

W tym celu w obiekcie `params` dodaj pustą tablicę `progress`. W funkcji `playerMove`, która jest uruchamiana po każdym ruchu gracza, wstawiaj do tablicy `params.progress` nowy obiekt zawierający wszystkie dane niezbędne do późniejszego wypełnienia tabeli.

Po zakończeniu gry, na podstawie obiektów w tablicy `params.progress` wygeneruj tabelę i wstaw ją do modala razem z komunikatem o zwycięzcy gry.

Etap 6* — dla chętnych - modal nowej gry

Ostatni etap jest "z gwiazdką", ponieważ wymaga poszukania dodatkowych informa
w internecie.

Twoim zadaniem w tym etapie jest zmienić działanie guzika "New game". Obecnie wyświetla on prompt pytający o liczbę wygranych rund, która ma oznaczać wygraną grę.

Stwórz nowy modal, który ma być otwierany po kliknięciu guzika "New game" zamiast prompta. W tym modalu chcemy mieć:

1. pole tekstowe (`<input type="text">`) na imię gracza,
2. pole tekstowe (`<input type="number">`) na ilość wygranych rund, która oznacza wygranie gry,
3. guzik "Start".

Tylko po kliknięciu guzika "Start" ma rozpocząć się nowa gra. Wykorzystaj limit wygranych rund tak jak do tej pory, a dodatkowo użyj imienia gracza na swojej stronie (np. jeśli imię to John, to zamiast "You played ROCK" ma być "John played ROCK").

Do wykonania tego etapu musisz:

- znaleźć w internecie informację, w jaki sposób pobierać wartość z pola tekstowego,
- dodać osobną funkcję wykonywaną po kliknięciu guzika "Start".

Nie bój się eksperymentować - a jeśli to zadanie pójdzie Ci zbyt szybko, zawsze możesz je rozbudować... ;)

Powodzenia!

Podgląd zadania

Wyślij link

Regulamin

Polityka prywatności