Witaj w drugim tygodniu Bootcampu!

Witaj w kolejnym tygodniu zmagań! Mam nadzieję, że poprzedni był dla Ciebie ciekawy i zawierał przydatne informacje. W tym tygodniu poznasz nieco bardziej zaawansowane mechanizmy języka JavaScript, które pozwolą Ci go nie tylko lepiej wykorzystać, ale przede wszystkim lepiej zrozumieć. Dowiesz się na czym polega dziedziczenie prototypowe, a także poznasz najważniejsze koncepcje programowania zorientowanego obiektowo. Będziesz pracować również z Ajaxem i formatem wymiany danych JSON, a na koniec poznasz kilka dobrych praktyk pracy z kodem JavaScript.

Grzegorz Róg

eduweb.pl

--- Zadania na tydzień 2 ---

Wszystkie prace, jak poprzednio, umieść w serwisie GitHub w osobnych repozytoriach.

1. Polyfill metody repeat dla String

W standardzie **EcmaScript 2015** pojawiła się nowa metoda dostępna na obiektach typu **String** o nazwie **repeat**. Jej użycie wygląda następująco:

```
“hej ”.repeat(3) // zwraca “hej hej hej ”
```

Metoda ta jest dostępna we wszystkich nowoczesnych przeglądarkach internetowych, ale aby poćwiczyć rozszerzanie wbudowanych typów, utwórz jej **polyfill**. W kodzie sprawdź najpierw czy taka metoda już w przeglądarce została zaimplementowana, a jeśli nie, to dopisz własną funkcję, która będzie mogła być na dowolnym stringu wywołana w podany wyżej sposób. Podpowiedź: rozszerzaj prototyp konstruktora **String**. Przy sprawdzeniu czy taka metoda już istnieje, w nowoczesnych przeglądarkach otrzymasz odpowiedź pozytywną. Jeśli zatem napiszesz odpowiedni warunek, to nie będzie można przetestować Twojej metody. Z tego powodu, zamiast **repeat** możesz ją nazwać **repeatt**.

2. Dziedziczenie z klasy EventEmitter

Przygotowany pod adresem http://pastebin.com/YEBncx0d kod zmodyfikuj tak, aby obiekty tworzone z klasy **Database** mogły korzystać z wszystkich metody klasy **EventEmitter**. Na chwilę obecną, podany kod wygeneruje błąd, gdyż klasa **Database** nie zawiera metody **on** oraz **emit**. Skorzystaj z dziedziczenia prototypowego aby klasą nadrzędną dla **Database** stała się klasa **EventEmitter**.

3. Ajaxowy polyfill dla funkcji fetch

Napisz **polyfill** dla funkcji fetch (nie będziemy się tutaj trzymać dokładnie tego, w jaki sposób ona działa, stworzysz jedynie prostą jej wersję). Wykorzystaj obiekt **XMLHttpRequest** w ten sposób, aby docelowo korzystanie z funkcji **fetch** wyglądało następująco:

```
fetch("url", function(data) {

    console.log("Sukces");

    console.log(data);

}, function(err) {

    console.log("Wystąpił błąd!");

    console.log(err);

});
```

a zatem jako pierwszy argument przekazujemy adres **URL** (wyślij pod niego zapytanie **GET**), jako drugi funkcję, którą należy wykonać **jeśli wszystko się powiedzie** (przekaż jej pobrane dane), a jako trzeci funkcję, która wykona się **na wypadek błędu** (przekaż jej obiekt z błędem lub komunikat tekstowy). W nowoczesnych przeglądarkach istnieje już funkcja **fetch**, a zatem aby jej nie nadpisywać, możesz nadać jej inną nazwę. Jako adres URL, z którego pobierane bedą dane, możesz wykorzytać https://jsonplaceholder.typicode.com/users

4. Prywatność za pomocą domknięcia

Dokończ pisanie przygotowanego pod adresem http://pastebin.com/aNKyCt3N kodu tak, aby nie generował błędów. Stwórz metody **get** oraz **set** nie korzystając z prototypów. Istotą działania funkcji **createData** jest zwrócenie obiektu, który zawierał będzie metody **get** oraz **set**. Metoda **get** powinna przyjmować klucz, np. “name” oraz zwracać wartość np. **data[“name”]**, natomiast metoda **set** powinna przyjmować klucz i wartość, sprawdzać czy oba te parametry zostały podane, a następnie powinna ustawiać np. **data[“name”] = “Janek”**.

5. Wrapper Toggler dla elementów z drzewa DOM

Do przygotowanego pod adresem http://pastebin.com/hUK5tnh3 kodu dodaj konstruktor (klasę) o nazwie **Toggler**. Przy tworzeniu nowych jej instancji z użyciem słowa kluczowego **new** (jak możesz zobaczyć w przygotowanym kodzie) przekazywać będziemy selektor. Za jego pomocą należy znaleźć na stronie odpowiedni element (skorzystaj z metody **document.querySelecto**r) i zapisać go w nowo stworzonym obiekcie. Następnie dodaj 3 metody. Pierwsza z nich o nazwie **getElem** powinna po prostu zwrócić znaleziony wcześniej element. Metoda **show** i **hide** powinny kolejno **pokazywać** i **ukrywać** element. Jeśli wszystko wykonasz poprawnie, kod który został już napisany powinien działać bez żadnych modyfikacji. Zauważ, że do elementu o identyfikatorze **button** zostało przypisane zdarzenie kliknięcia. Taki element musisz wstawić na stronę, podobnie jak i element, którego selektor zostanie przekazany przy tworzeniu nowego obiektu klasy **Toggler**.

W JavaScript prawie wszystko jest obiektem

Język JavaScript jest na wiele sposobów wyjątkowy. Jedną z jego głównych cech jest fakt, iż prawie wszystko jest obiektem. Nawet jeżeli już swobodnie poruszasz się po tym języku, możesz nie zauważać, kiedy masz do czynienia z obiektami. Świadomość obiektowości niesie ze sobą wiele pozytywnych konsekwencji. Dzięki niej możesz dynamicznie rozszerzać dowolne obiekty, przechowywać z nich dane lub dodawać do nich metody.

Programowanie Zorientowane Obiektowo

OOP to z języka angielskiego skrót od **Object Oriented Programming**. Oznacza to **Programowanie Zorientowane Obiektowo** lub jak się potocznie zwykło mówić "**programowanie obiektowe**". Sercem tego zjawiska w języku JavaScript jest mechanizm prototypów. Okazuje się bowiem, że każdy obiekt, z którym pracujemy, jest podłączony do pewnego łańcucha prototypów. W ten sposób tworzona jest hierarchia obiektów, które współdzielą pomiędzy sobą zapisane w sobie dane oraz metody. Zrozumienie tego mechanizmu pozwoli Ci tworzyć własne typy obiektów (klasy), a następnie ich instancje (konkretne obiekty).

Ajax i JSON

Od “Ajaxowego boom’u”, który miał miejsce w 2005 roku wiele się zmieniło. Wcześniej strony internetowe, aby wyświetlić jakąś nową zawartość, potrzebowały całkowitego przeładowania. Dzisiaj w wielu przypadkach jest zupełnie inaczej. Dodajemy produkt do koszyka, a on pojawia się tam natychmiast. Wszystko to dzięki asynchronicznemu wywołaniu, które realizowane jest z poziomu kodu JavaScript. Skoro Ajax pozwala przesyłać dane do i z serwera, dobrze byłoby przesyłać je w jakimś powszechnie stosowanym formacie. I tak też jest! Kiedyś popularny był format **XML**, ale już od wielu lat króluje **JSON**. To lekki format wymiany danych, który jest naturalny dla języka JavaScript, ale jest także dobrze “rozumiany” przez technologie serwerowe, takie jak np. PHP. Dodawanie ajaxowych komponentów to stron internetowych to jednak nie wszystko. Dzięki rozwojowi licznych frameworków tworzonych w języku JavaScript, a także możliwościom Ajaxa, dzisiaj tworzone są bardzo zaawansowane aplikacje internetowe, które działają wprost w przeglądarce internetowej.