Technologie informacyjne
przed 12 wykładem

Andrzej Giniewicz

5.06.2024

Kolejna porcja materiału dotyczy języka ECMAScript, który zapewnia wysoką interaktywność stronom internetowym.

1 Historia i nazwa

Podczas wojny przeglądarek firma Netscape zastanawiała się, jak można sprawić, aby dać twórcom stron więcej możliwości tworzenia interaktywnych elementów, choćby jak tag `blink` służący do tworzenia migającego tekstu lub `marquee` do przesuwającego się tekstu. Tagi `blink` i `marquee` były problematyczne — jeden działał tylko w przeglądarce Netscape, drugi w Internet Explorer. Dodanie do przeglądarki języka, w którym to twórca strony mógłby zaprogramować takie lub inne elementy interaktywne i przesłać kod do osób odwiedzających stronę, stanowiłoby o wiele ogólniejsze rozwiązanie. Początkowo próbowano osadzić w przeglądarce język Java lub Scheme — w tamtych czasach jeden z nich był nowoczesnym językiem obiektowym, drugi językiem funkcyjnym z długą tradycją. Scheme dawał możliwość uruchamiania programów jako skrypty, więc etap kompilacji kodu w przeglądarce nie był konieczny. Java natomiast charakteryzowała się składnią przyjaźniejszą dla większej liczby programistów. Decyzją, jaką podjęto, to opracowanie nowego języka skryptowego, ze składnią zbliżoną do języka Java. Oprócz założenia, że składnia tego języka jest bardziej podobna do Java niż do Scheme, nowo-powstały język nie miał wspólnych korzeni z językiem Java. Wersja beta języka nazywała się LiveScript, jednak w 1995 roku przy wydaniu przeglądarki Netscape Navigator 2.0, zmieniono jego nazwę na JavaScript, aby skorzystać z popularności nazwy.

Stosując inżynierię wsteczną interpretera języka JavaScript firma Microsoft w 1996 roku stworzyła swoją wersję języka o nazwie JScript. Firma Netscape w tym samym roku przekazała specyfikację języka JavaScript organizacji standaryzującej ECMA (European Computer Manufacturers Association). Rok później — w 1997 roku — pojawiła się pierwsza wersja standardu ECMAScript, opracowanego na tej postawie. Choć Microsoft początkowo uczestniczył w standaryzacji, w momencie, gdy zaczął wygrywać pierwszą wojnę przeglądarek, wycofał się. W efekcie, choć JScript nie był zgodny ze standardem, był normą, ponieważ 95% użytkowników stron internetowych używało przeglądarki Internet Explorer z JScript. W 2005 roku pojawiła się technologia Ajax (Asynchronous JavaScript and XML), która

pozwalała ładować zasoby w tle, bez przeładowywania stron. Trzy lata później pojawiła się koncepcja sposobu łączenia o nazwie WebSocket, która pozwalała na komunikację w obie strony — teraz to serwer również mógł wysłać informacje bez zapytania, co okazało się bardzo pomocne przy implementacji wielu aplikacji, w szczególności komunikatorów działających przez strony www. Technologie te, dla których podstawą był JavaScript, okazały się na tyle wartościowe, że producenci przeglądarek zdecydowali się ponownie współpracować nad standardami, tworząc w 2009 roku wersję 5 standardu ECMAScript. Obecnie większość przeglądarek w pełni wspiera standard ECMAScript 5[1]. Inną technologią również powstałą w 2009 roku i silnie powiązaną z ECMAScript jest Node.js[2]. Jest to tak zwany *runtime*[3], służący do uruchamiania aplikacji w ECMAScript bez przeglądarki. Jest bardzo często używany na serwerach, dzięki czemu całość strony może być implementowana w jednym języku.

Technicznie, JavaScript jest jedną z implementacji standardu ECMAScript — podobnie jak JScript oraz wiele innych. Obecnie odchodzi się od nazwy JavaScript z powodów licencyjnych[4]. Jednym z warunków współpracy pomiędzy firmą Sun a Netscape było to, że przy zmianie nazwy, Sun będzie właścicielem nazwy, a Netscape otrzyma dożywotnią licencję na jej używanie. Obecnie ani jednej ani drugiej firmy już nie ma — Netscape od 2003, Sun od 2010. Firmę Sun wraz z prawami do nazwy JavaScript kupiła firma Oracle i jest dziś właścicielem nazwy. Znane są sytuacje zgłaszania przez firmę Oracle praw do nazw przez nich posiadanych, w tym zgłoszenia naruszenia znaku towarowego w stosunku do osoby tworzącej edytor HTML, CSS i JavaScript, która użyła „JavaScript" w nazwie. Z tego powodu, coraz częściej używa się nazwy standardu, a nie konkretnej implementacji. Niemniej jednak niekiedy można spotkać nazwę JavaScript w odniesieniu do języka ECMAScript, ze względu na jej popularność.

Kolejne wersje standardu ECMAScript są określane rokiem wydania. Od 2015 roku co rok ukazywał się nowy standard, aż do 2023 roku. Obecnie trwają prace nad ECMAScript 2024, zaplanowanym na lipiec tego roku. Równolegle rozpoczęły się prace nad standardem ECMAScript 2025. Funkcjonalność dostępna w nowych wersjach przeglądarek pokrywa większość standardu[5] — najnowsze wersje przeglądarek, na przykład Chrome 112, Firefox 116 oraz Opera 98, posiadają pełne wsparcie standardu ECMAScript do wersji 2024 włącznie[6], za wyjątkiem obsługi najnowszego standardu Unicode 15.1, który ukazał się w sierpniu 2023 roku[7].

[1] Porównanie wsparcia znajduje się na stronie `https://compat-table.github.io/compat-table/es5/`.

[2] `https://nodejs.org/`.

[3] Do innych należą na przykład Deno – `https://deno.com/` lub Bun – `https://bun.sh/`.

[4] Patrz `http://web.archive.org/web/20230220194838/https://www.techrepublic.com/article/why-its-finally-time-to-give-up-on-the-name-javascript/`.

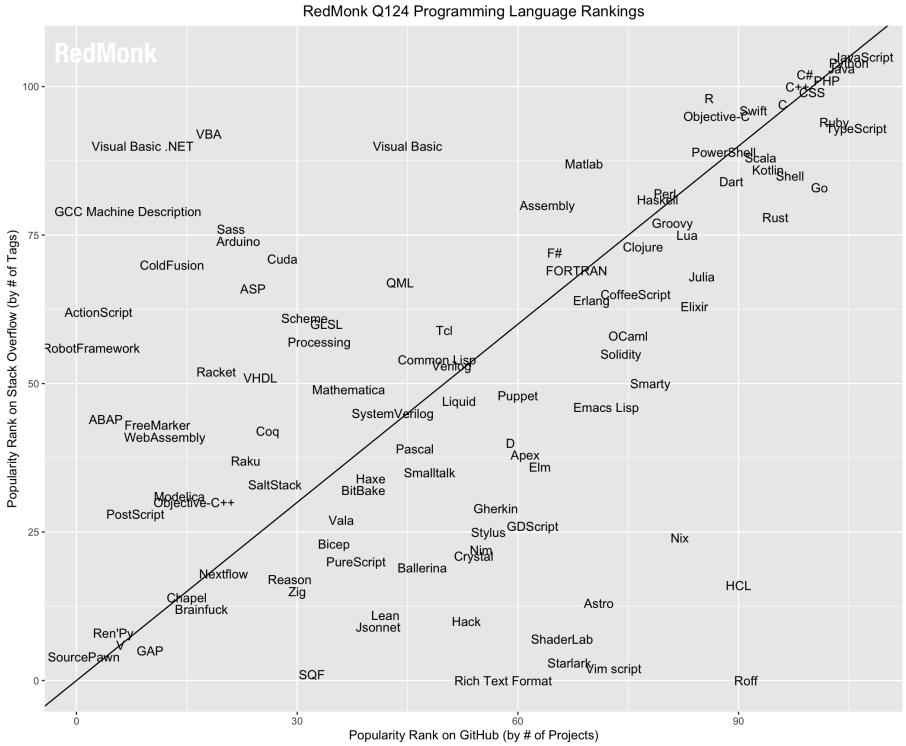
[5] Tabela z porównaniami znajduje się na `https://compat-table.github.io/compat-table/es2016plus/`.

[6] W teście sprawdzającym obsługę jednej z funkcjonalności z 2022 roku znajduje się błąd — z powodu pojawienia się nowej flagi `unicodeSets` opisanej w `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/unicodeSets`, wszystkie przeglądarki obsługujące flagę `unicodeSets` są nieprawidłowo klasyfikowane, jako nieobsługujące flagi `hasIndices`.

[7] Patrz `http://blog.unicode.org/2023/09/announcing-unicode-standard-version-151.html`.

Do 2019 roku HTML, CSS i ECMAScript stanowiły trzon podstawowych technologii sieciowych w dzisiejszych czasach. W grudniu 2019 roku do zestawu dołączył WebAssembly[8], uzupełniając zbiór technologii dostępnych w przeglądarkach. WebAssembly pozwala na uruchamianie programów napisanych w innych językach, kompilowanych do przenośnej formy. Funkcje napisane w WebAssembly mogą być uruchomione z poziomu JavaScript, więc pozwala to przyśpieszyć wykonywanie niektórych fragmentów kodu. Jest to najnowszy dodatek do środowiska, pozwalający na tworzenie kodu dla przeglądarek w językach takich, jak C, C++, Go lub Rust.

ECMAScript jest obok Pythona jednym z najpopularniejszych języków programowania



Źródło: https://redmonk.com/sogrady/2024/03/08/language-rankings-1-24/.

Niestety jego historia i rozwój spowodował, że ma opinię języka, który nie jest najlepiej zorganizowany.

Źródło: https://www.movereem.nl/JavaScript-sources.

[8] https://www.w3.org/TR/wasm-core-1/.

2 ECMAScript na nasze potrzeby

My podczas zajęć zajmiemy się tylko niewielkim podzbiorem języka ECMAScript, który będzie wykorzystywany w przeglądarce, do tworzenia podstawowych interakcji. Kod załączamy do strony tagiem `script`, umieszczając go bezpośrednio

```
<script>
kod
</script>
```

lub z zewnętrznego pliku

```
<script src="plik.js"></script>
```

Znacznik `script` może znajdować się w `head` lub `body`. Może też posiadać dodatkowe puste atrybuty, które sterują tym, kiedy zostanie uruchomiony:

- `async` — skrypt będzie ładował się w tle i nie będzie przeszkadzał stronie. Nie powinniśmy używać tego rodzaju ładowania, jeśli skrypt modyfikuje treść strony lub jest biblioteką, z której korzystają inne skrypty;
- `defer` bez `async` — skrypt uruchamia się po tym, jak wczytana zostanie cała strona, również ze skryptami umieszczonymi w `body`. Defer działa tylko, gdy używamy atrybutu `src`;
- brak `defer` i `async` — skrypt uruchamia się w momencie napotkania tagu i blokuje wczytanie strony do momentu zakończenia działania skryptu.

Zaleca się ładowanie skryptów w tagu `head` i użycie `defer` dla tych skryptów, które mogą lub muszą poczekać na wyświetlenie treści strony.

Oprócz uruchamiania skryptów ważnym elementem integracji języka ECMAScript z HTML są zdarzenia, które pozwalają uruchomić kod w odpowiedzi na różne sytuacje. Rozważymy trzy sposoby obsługi zdarzeń:

- dopisanie obsługi zdarzenia do tagu. Najpopularniejsze zdarzenia, to `onclick` oraz `onload`. Przykładowo:

  ```
  <p onclick="funkcja();">Po kliknięciu tekstu, uruchomi się funkcja.</p>
  ```

  Listę zdarzeń znajdziemy na przykład na stronie `http://www.w3schools.com/jsref/dom_obj_event.asp`;
- wykorzystanie biblioteki jQuery[9], która pozwala dodawać obsługę zdarzeń za pomocą selektorów. W przykładzie poniżej funkcja zostanie uruchomiona po kliknięciu dowolnego akapitu

[9] `https://jquery.com/`.

```
$(”p”).click(funkcja);
```

- wykorzystanie nowych możliwości ECMAScript i ręczne budowanie obsługi zdarzeń bez jQuery. W przykładzie poniżej funkcja jest doczepiona pod zdarzenie „click” dla obiektu o konkretnym `id=”lead”`

  ```
  document.getElementById(’lead’).addEventListener(’click’, funkcja);
  ```

  O tym jak nowe możliwości ECMAScript pozwalają zastąpić jQuery, możemy przeczytać na stronie `https://love2dev.com/blog/jquery-obsolete/`,

- wykorzystanie biblioteki Alpine.JS[10] pozwalającą na programowanie reaktywne

  ```
  <div x-data=”{ open: false }”>
      <button @click=”open = true”>Rozwiń</button>

      <span x-show=”open”>
          Początkowo ukryta zawartość
      </span>
  </div>
  ```

- wykorzystanie biblioteki HTMX[11], która sama będąc biblioteką napisaną w ECMAScript, rozbudowuje HTML jako hiper-medium[12] i pozwala na dodawanie interaktywności bez pisania kodu w ECMAScript — w przypadku HTMX reakcja na przycisk nie jest generowana przez przeglądarkę, tylko przez serwer.

Wszystkie te sposoby mają woje wady i zalety, które omówimy bliżej na wykładzie. Do zajęć pomimo dostępnych alternatyw i tego, że jest to wybór cieszący się złą sławą, wybraliśmy bibliotekę jQuery — powody były następujące:

- chcieliśmy wybrać sposób, który pozwala napisać interakcję na poziomie przeglądarki, a nie serwera, co wykluczyło bibliotekę HTMX,
- chcieliśmy wybrać sposób, w którym nie musimy poznawać nowego paradygmatu programowania, co wykluczyło bibliotekę Alpine.JS,
- chcieliśmy wybrać sposób, w którym kod będzie zwięzły i czytelny, co wykluczyło podejście korzystające rejestrację zdarzeń przez ECMAScript za pomocą `getElementById`,

[10] `https://alpinejs.dev/`.

[11] `https://htmx.org/`.

[12] O HTMX oraz jak wykorzystać go w bibliotece Flask, można przeczytać w ogólnodostępnej książce Hypermedia Systems – `https://hypermedia.systems/`.

- chcieliśmy wybrać sposób, w którym mamy możliwość podłączenia tego samego zdarzenia pod wiele elementów HTML, co wyklucza pierwsze podejście, korzystające z czystego ECMAScript oraz zdarzeń typu `onclick`.

To z wypisanych rozwiązań pozostawia jQuery, które jest używane na blisko połowie stron internetowych na świecie[13].

3 Kolejny język programowania

Jako osobom, które znają już Pythona, poznały podstawy BASHa i LaTeXa, nie będziemy tłumaczyć po raz kolejny podstawowych elementów składni, takich jak pętle oraz instrukcje warunkowe. Ich działanie jest niemal identyczne z większością poznanych dotąd języków. W sekcji tej przedstawimy jedynie szybki przegląd najważniejszych elementów składni, ze zwróceniem uwagi na nietypowe elementy.

3.1 Komentarze

Komentarze zaczynamy od `//` lub umieszczamy w `/* ... */`. Poniżej przykład obu wariantów komentarzy.

```
// na jedną linię

/* na wiele
linijek */
```

3.2 Literały

ECMAScript ma możliwość zapisywania literałów liczbowych zmiennoprzecinkowych, zgodnych ze standardem IEEE-754. Przykładowe wartości to

```
3
1.5
Infinity
NaN
```

Istnieją również literały logiczne

```
true
false
```

oraz napisowe w pojedynczych lub podwójnych cudzysłowach

[13] Patrz `https://trends.builtwith.com/javascript/javascript-library/traffic/Entire-Internet`.

```
'napis'
"napis"
```

Specjalnymi literałami są `null` oraz `undefined`. `undefined` jest zarezerwowane dla zmiennych, które jeszcze nie mają przypisanej wartości oraz dla nieistniejących atrybutów obiektów. Wartość `null` jest natomiast określona i oznacza pustkę. Jeśli tworzylibyśmy aplikację rozwiązującą równania, mogłaby zwracać liczbę, gdy rozwiązanie istnieje, wartość `null`, gdy rozwiązania nie ma oraz `undefined`, gdy nie udało się policzyć i sprawdzić, czy rozwiązanie istnieje, czy nie. Rozróżnienie tych wartości jest niekiedy przydatne, ale może sprawiać problemy na początkowych etapach przygody z ECMAScript.

3.3 Operatory

W ECMAScript mamy operatory arytmetyczne, między innymi

```
1 + 2
1 - 2
1 * 2
1 / 2
1 % 2
```

operatory logiczne, oznaczające odpowiednio negację, iloczyn i sumę logiczną

```
!false
true && true
true || false
```

Mamy również dostęp do operatorów relacji. Na szczególną uwagę zasługują operatory porównania, która mają dwa warianty — podwójne oraz potrójne.

```
1 < 2
1 <= 2
1 == 2
1 != 2
1 === 2
1 !== 2
```

Różnica pomiędzy podwójnymi a potrójnymi operatorami jest taka, że podwójny sprawdza wartości, podczas gdy potrójny sprawdza również typy. Jest to rozróżnienie podobne, choć nie dokładnie takie samo, jak w przypadku operatorów `==` oraz `is` w Pythonie. Działanie operatorów w połączeniu z typem `undefined` może być zastanawiające. Poniżej kilka przykładów.

```
1 == "1";   // true
1 === "1";  // false
null == undefined;   // true
null === undefined;  // false
```

Więcej zastanawiających zachowań ECMAScript możemy znaleźć w Internecie[14].

3.4 Zmienne

ECMAScript rozpoznaje wielkość liter. Oznacza to, że zmienna `punkt` i `Punkt` to dwie różne nazwy. Jeśli chcemy ustawić zmienną globalną na jakąś wartość, możemy to zrobić pisząc

```
zmienna = wartość;
```

Zwróćmy uwagę, że linie w ECMAScript kończymy średnikiem. Przypisanie takie będzie widoczne w całym programie. Jeśli chcemy zadeklarować zmienną lokalną, piszemy

```
var zmienna_lokalna;
```

Wartością tej zmiennej jest `undefined`, do momentu aż nie podstawimy pod nią jakiejś wartości, czyli jej nie zdefiniujemy. Deklarację i definicję zmiennej możemy połączyć lub umieścić w oddzielnych liniach.

```
var x;
x = 7;

var y = 7;
```

3.5 Złożone typy danych

ECMAScript obsługuje tablice i słowniki. Tablice definiujemy w nawiasach kwadratowych. Odwoływać się do elementów możemy za pomocą indeksów rozpoczynających się od zera.

```
var tablica = ["1", 2, true];
tablica[1] == 2;
```

Tablice są obiektami, więc mają metody, na przykład metodę `push` dodającą nowy element na koniec listy. Mają również atrybuty, takie jak `length`.

[14]Na przykład na stronie `https://github.com/denysdovhan/wtfjs/blob/master/README-pl-pl.md`.

```
tablica.push(4);
tablica.length == 4;
```

Słowniki również są obiektami. Możemy odwoływać się do kluczy za pomocą atrybutów lub nawiasów kwadratowych, co jest przydatne, gdy klucz ma w sobie nielegalny znak dla atrybutów, na przykład spację.

```
var obiekt = {klucz: ”wartość”, ”inny klucz”: 2};
obiekt.klucz == ”wartość”;
obiekt[”inny klucz”] == 2;
```

Obiekty w ECMAScript, w tym słowniki, są otwarte, to znaczy, że w każdej chwili możemy dodać nowy atrybut. Ciekawostką jest to, że odwołanie się do nieistniejącego atrybutu nie jest błędem, tylko zwraca wartość `undefined`.

```
obiekt.x = 3;
obiekt.x == 3;
obiekt.y == undefined;
```

4 Instrukcje sterujące

Najprostszą instrukcją sterującą jest instrukcja warunkowa `if`.

```
if (warunek) {
    // kod
} else if (warunek) {
    // kod
} else {
    // kod
}
```

Język posiada również pętlę `while` oraz `do-while`

```
while (warunek) {
    // kod
}

do {
    // kod
} while(warunek);
```

Pętle te różnią się tym, kiedy sprawdzany będzie warunek kontynuacji — w pierwszym przypadku jest on przed uruchomieniem, w drugim po uruchomieniu. Oznacza to, że pętla `do-while` wykona się przynajmniej raz. Istnieje też pętla `for`, na przykład

```
for (var i=1; i<=10; i++) {
    // kod
}
```

wykonująca się dla *i* od 1 do 10 co 1. Pętla ta posiada również wariant dla tablic, przypominający pętlę `for` z Pythona.

```
for (var x in tablica) {
    z += x;
}
```

5 Funkcje

Funkcje definiujemy za pomocą słowa kluczowego `function`. Wartość zwracamy za pomocą słowa kluczowego `return`.

```
function nazwa(argumenty){
    // kod
    return wynik;
}
```

Aby użyć funkcji, musimy użyć jej nazwy i podać odpowiednie argumenty.

```
nazwa(argumenty) == wynik;
```

Istnieje możliwość stworzenia funkcji anonimowej, czyli funkcji bez nazwy. Są one podobne do wyrażeń lambda w Pythonie. Piszemy

```
function (argumenty) { kod }
```

najczęściej funkcja taka jest od razu przekazana jako argument do innej funkcji, na przykład do funkcji obsługującej zdarzenie.

Funkcje anonimowe bez argumentów mają też częste zastosowanie w opóźnieniu wykonania. Ponieważ przeglądarki wczytują kod kawałek po kawałku, może się zdarzyć, że wykonają fragment kodu i będą musiały poczekać, aż wczyta się reszta. Aby temu zaradzić, można spotkać następujący zapis

```
(function() {
 // kod
})();
```

Co się tu dzieje? Mamy anonimową definicję funkcji bezargumentowej w nawiasie — zaraz za nim, jest pusty nawias, który powoduje uruchomienie anonimowej funkcji. W efekcie kod w klamerkach zacznie się wykonywać, dopiero gdy przeglądarka dojdzie do nawiasów uruchamiających funkcję, czyli gdy całość kodu zostanie wczytana.

Funkcje bez argumentów możemy też uruchamiać po jakimś czasie

```
setTimeout(nazwa, czas w ms);  // uruchom po czasie
```

lub co jakiś czas, przykładowo co sekundę.

```
setInterval(nazwa, czas w ms); // uruchamiaj co czas
```

Przykładowo, aby uruchomić kod co sekundę, napiszemy

```
setInterval(function() {
 // kod
}, 1000);
```

5.1 jQuery

Aby załadować jQuery, należy dodać odpowiedni skrypt. Tag `script` pobieramy ze strony `https://code.jquery.com/`. Wybieramy najnowszą wersję[15], na moment pisania jest to 3.7.1. Klikamy przycisk podpisany jako `minified` i kopiujemy kod. Uzyskamy kod podobny do

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"
     integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
        crossorigin="anonymous"></script>
```

Skrypt taki umieszczamy zwykle na końcu tagu `body` po treści. Przykładowa strona to

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="utf-8">
  <title>Test jQuery</title>
  <style>
#hidden { display: none; font-size: 300%; }
  </style>
</head>
```

[15]Wersja 4.0.0 beta pojawiła w lutym tego roku, patrz `https://blog.jquery.com/2024/02/06/jquery-4-0-0-beta/`.

```
<body>

  <h1 id="welcome">Hello</h1>
  <p id="hidden">World!</p>

  <script src="https://code.jquery.com/jquery-3.7.1.min.js"
     integrity="sha256-/JqT3SQfawRcv/BIHPThkBvsOOEvtFFmqPF/lYI/Cxo="
        crossorigin="anonymous"></script>
  <script>

$("#welcome").on('click', function(e) {
    $("#hidden").slideToggle();
});
$("#hidden").on('click', function(e) {
    $(this).html("ECMAScript!");
});

  </script>
</body>
</html>
```

Działanie tego skryptu można zobaczyć po zapisaniu go w HTML lub w wygodnym narzędziu do demonstracji przykładów — wejdź na stronę `https://jsfiddle.net/bmry49je/`, kliknij napis „Hello", a następnie „World". Postaraj się zrozumieć kod, który jest odpowiedzialny za zachowanie strony. Jeśli masz problem, spróbuj przeczytać odpowiednie fragmenty w dokumentacji jQuery:

- `https://api.jquery.com/jquery/`,
- `https://api.jquery.com/on/`,
- `https://api.jquery.com/slidetoggle/`,
- `https://api.jquery.com/html/`.

5.2 Więcej o ECMAScript

Bardzo bogate materiały o ECMAScript znajdują się na stronie `https://developer.mozilla.org/en-US/docs/Web/JavaScript`, czyli tam, skąd ECMAScript się wywodzi. Jeśli ktoś woli materiały typu przewodnikowego, sporo sprawdzonych informacji jest na stronie `https://www.w3schools.com/js/` oraz `https://www.w3schools.com/jquery/`. Nieocenionym narzędziem do pracy z kodem będzie też walidator `https://esprima.org/demo/validate.html`.

5.3 Biblioteki

Oprócz jQuery istnieje wiele bibliotek. Na szczególną uwagę zasługują:

- `https://www.mathjax.org/` — biblioteka do wstawiania wzorów w LaTeXu,
- `https://tikzjax.com/` — biblioteka do wstawiania rysunków w TikZ,
- `https://jsxgraph.org/` — biblioteka do interaktywnych wykresów,
- `https://d3js.org/` — biblioteka do wizualizacji danych,
- `http://threejs.org/` — biblioteka do grafiki 3D.

Postaraj się obejrzeć przykłady ze stron z wymienionymi bibliotekami.