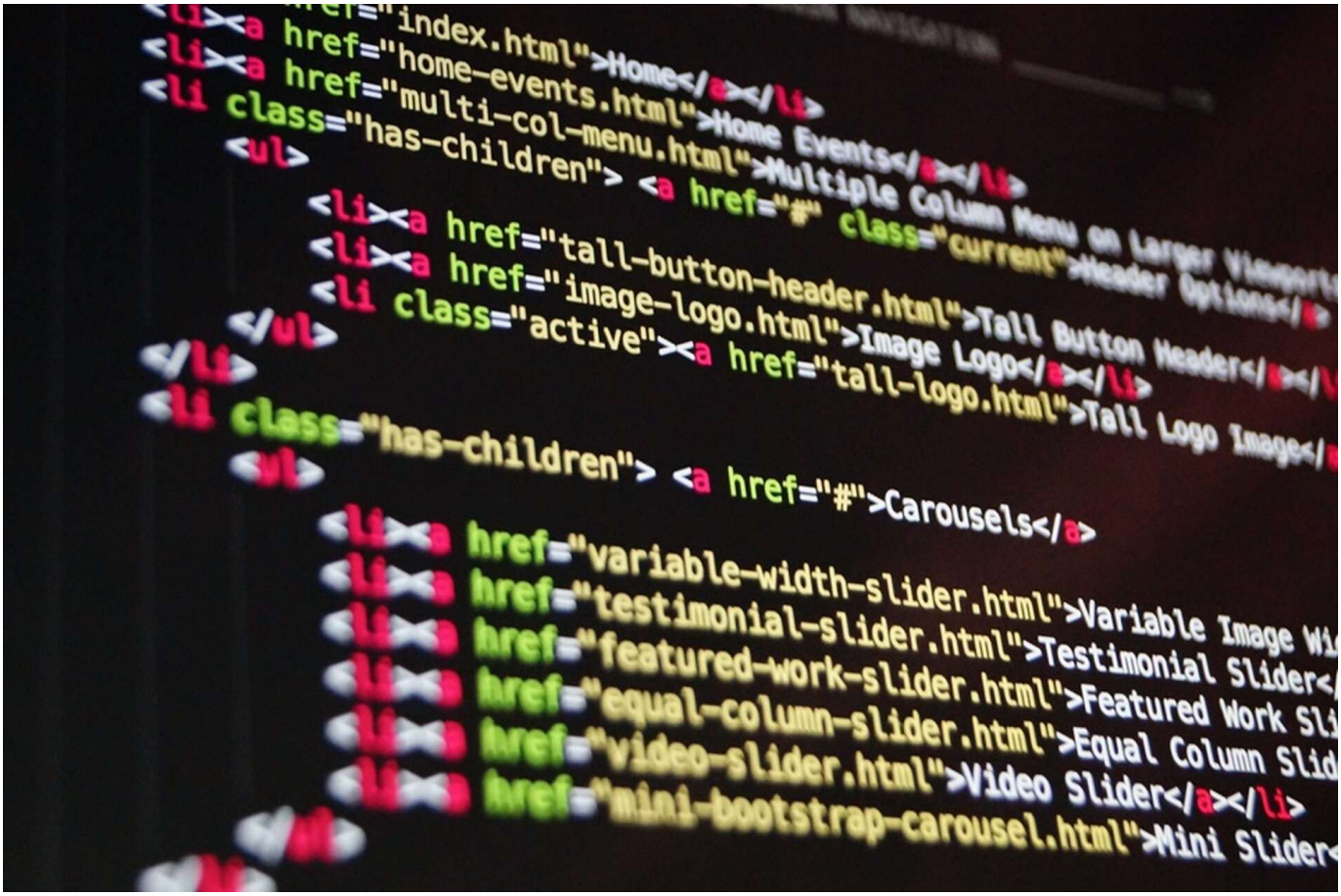


Podstawy języka JavaScript

- Wprowadzenie
- Przeczytaj
- Prezentacja multimedialna
- Sprawdź się
- Dla nauczyciela

Źródło: domena publiczna.

JavaScript to nie tylko język opisowy, taki jak HTML czy CSS. Jest to pełnoprawny, skryptowy język programowania, w którym możemy zastosować cały repertuar klasycznych konstrukcji językowych, takich jak: instrukcje warunkowe, pętle, zmienne, tablice, instrukcje wyboru, własne funkcje, klasy, obiekty itd.

Na początku przygody z tworzeniem stron internetowych języka JavaScript używamy najczęściej do ulepszenia interfejsu witryny, wzbogacając ją o dodatkowe funkcjonalności niedostępne w HTML i CSS. Skrypty pozwalają nam tworzyć animowane galerie zdjęć, wyskakujące panele nawigacyjne, interaktywne menu, a także umożliwiają przetworzenie wprowadzonych w formularzu danych czy wykonywanie obliczeń matematycznych.

W tym e-materiale dowiesz się także, jak wygląda ścieżka rozwoju programisty JavaScript, i poznasz współczesne technologie używające tego języka.

Twoje cele

- Porównasz zastosowanie współczesnych technologii wykorzystujących język JavaScript.
- Dobierzesz sposób dołączenia skryptu JavaScript do kodu źródłowego HTML witryny.
- Zdefiniujesz zmienną, stałą, uchwyt obiektu, własną funkcję oraz obsługę zdarzenia.

- Zastosujesz operatory używane w JavaScript.
- Sklasyfikujesz podstawowe rodzaje okien dialogowych w przeglądarce internetowej.

Przeczytaj

Zacznijmy od omówienia podstawowych mechanik oraz konstrukcji językowych, które warto poznać na początku przygody z językiem JavaScript.

Dołączanie skryptu do kodu HTML witryny

Jak rozpocząć pisanie skryptów w języku JavaScript? Należy pamiętać, że zapisany w nim kod źródłowy należy oddzielić od zapisów wykonanych w językach opisowych (HTML, CSS). **Zawartość** strony (HTML) jest czymś innym niż jej **wygląd** (CSS) oraz **mechanika** działania (JavaScript). Przeglądarka internetowa musi wiedzieć, który kod jedynie opisuje elementy na stronie, a które zapisy należy potraktować jako skrypt do wykonania.

Skrypt napisany w języku JavaScript można osadzić wewnątrz kodu HTML tworzonej witryny internetowej na dwa sposoby: bezpośrednio w dokumencie oraz z zewnętrznego pliku.

Skrypt umieszczony bezpośrednio w dokumencie

Możliwe jest umieszczenie kodu źródłowego skryptu wprost w dokumencie HTML - wpisujemy go wtedy pomiędzy dodatkowe znaczniki `<script>` i `</script>`:

```
<script>
   alert("Witaj Świecie!");
   //kod JS znajduje się pomiędzy znacznikami script
</script>
```

To właśnie dzięki nim przeglądarka potrafi odróżnić fragment kodu napisanego w JavaScript od zapisów, które należy interpretować jako język HTML.

W samym skrypcie użyto funkcji `alert()`, która wyświetla przesłany jej pomiędzy nawiasami tekst w małym oknie dialogowym generowanym przez przeglądarkę:

Źródło: domena publiczna.

Wygląd tego okna może różnić się w zależności od użytej przeglądarki. Wszystkie rodzaje okien dialogowych poznamy w dalszej części tego e-materiału.

Skrypt osadzony z zewnętrznego pliku

Możemy również umieścić kod źródłowy skryptu w zewnętrznym pliku z rozszerzeniem `.js` oraz zaimplementować go w dokumencie HTML:

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

Zaletą tego rozwiązania jest wyraźne oddzielenie znaczników HTML od kodu JavaScript, do czego powinien zawsze dążyć programista odpowiedzialny za front-end.

Atrybut `async`

W wypadku osadzenia skryptu w dokumencie zewnętrznym możemy zastosować atrybut `async` wprowadzony w standardzie HTML5.

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

Zachowaniem domyślnym przeglądarki, gdy napotka zewnętrzny skrypt na stronie, jest parsowanie i wykonanie skryptu natychmiast w linii dołączenia, zanim kontynuowane będzie czytanie dalszych linii źródła.

Obecność atrybutu `async` zmienia tę sytuację – parsowanie HTML nie zostaje zatrzymane, a pobieranie skryptu odbywa się w tym samym czasie co parsowanie kolejnych linii HTML. Dopiero kiedy skrypt zostanie już w całości pobrany i przygotowany do wykonania, przeglądarka zatrzyma przetwarzanie kodu HTML i wykona skrypt.

Atrybut `defer`

Natomiast w sytuacji, w której nie ustawimy atrybutu `async`, a wprowadzimy atrybut o nazwie `defer`:

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

przeglądarka wykona zewnętrzny skrypt dopiero wtedy, gdy kod źródłowy witryny zostanie w całości wczytany.

> **Ważne!**

> Istotna różnica między tymi atrybutami polega na tym, że skrypty z atrybutem `defer` będą uruchamiane w kolejności, w jakiej zostały umieszczone w dokumencie, tymczasem skrypty z atrybutem `async` podlegają kolejności wczytywania się – jako pierwszy uruchomi się ten skrypt, który zostanie wczytany jako pierwszy.

Deklarowanie zmiennych

Zmienna to (najprościej mówiąc) pojemnik na dane – możemy w niej przechować np. liczbę, napis, wartość logiczną. Stosujemy ją, gdy potrzebujemy zapamiętać w skrypcie jakąś wartość. Informacja ta przechowywana jest w pamięci RAM komputera.

Zmienne w języku JavaScript deklarujemy z użyciem klauzuli `var` (ang. *variable* – zmienna), po której wpisujemy nazwę zmiennej:

```
<script>
  var ile_jablek = 12;
</script>
```

Tworząc nazwę zmiennej, nie możemy:

- rozpocząć jej od cyfry (przez wzgląd na szesnastkowy zapis liczb),
- użyć spacji; jeżeli chcemy, aby nazwa składała się z kilku słów, stosujemy znak podkreślenia (np. `ile_jablek`, a nie: `ile-jablek`),
- wykorzystywać słów kluczowych języka JavaScript.

Powinniśmy także unikać polskich znaków diakrytycznych (choć nie jest to zakazane). Warto natomiast stosować angielskie nazewnictwo zmiennych, co usprawnia pracę w międzynarodowych zespołach programistów.

W języku JavaScript możemy zadeklarować zmienną, używając klauzuli `let` (ang. zezwalać):

```
<script>
  let liczba = 20;
</script>
```

Ten sposób pozwala rozwiązać pewien problem z zasięgami zmiennych stworzonych za pomocą klauzuli `var`. Otóż pojemniki stworzone z użyciem klauzuli `let` mają tzw. **zasięg blokowy**, czyli pozostają widoczne jedynie w obrębie konstrukcji językowej, w której się znajdują. Natomiast zmienne tworzone z użyciem klauzuli `var` pozostają zawsze widoczne lokalnie w całej funkcji.

Na początkowym etapie nauki ta różnica nie ma dużego znaczenia. Z czasem jednak zaczniemy zauważać potrzebę unikania zjawiska tzw. hoistingu zasięgu zmiennych.

Stałe

Oprócz zmiennych, w których przechowywane wartości mogą się dynamicznie zmieniać w trakcie wykonania skryptu, w języku JavaScript możemy też zadeklarować tzw. **stałą**. Jest to taki pojemnik na dane, którego zawartość nie ulega zmianom. Stałą można stworzyć z użyciem klauzuli `const` (z ang. *constant* – stały, niezmienny):

```
<script>
  const PI = 3.141592653;
  PI = 14.3; // błąd – zmiana wartości stałej!
</script>
```

Wymienione już reguły nazewnictwa zmiennych obowiązują także dla stałych. Należy jednak pamiętać, że stała nie może mieć identycznej nazwy jak istniejąca w tym samym zasięgu zmienna lub funkcja.

Uchwycenie elementu witryny

Wybrany element struktury dokumentu HTML, który ma ustawioną wartość atrybutu `id` (unikalny identyfikator), można **uchwycić**, czyli umożliwić pracę z tym obiektem w języku JavaScript. Używamy w tym celu metody `document.getElementById()`, co czytamy w następujący sposób: „Z całego dokumentu uchwyć do edycji obiekt o następującym id".

HTML:

```
<input type="text" id="pole">
```

JavaScript:

```
<script>
  var napis = document.getElementById("pole").value;
</script>
```

Uchwyt pozwala uzyskać łatwy dostęp do atrybutów i metod obiektu. Możemy je w ten sposób odczytywać i modyfikować. W przywołanym przykładzie uchwycono zdefiniowane

w HTML pole edycyjne o identyfikatorze „pole”, po czym odczytano wartość jego atrybutu `value`. Atrybut ten przechowuje napis aktualnie znajdujący się w polu tekstowym.

W zmiennej `napis` znajdzie się więc tekst pobrany ze wskazanego atrybutem `id` pola edycyjnego w dokumencie. Co ważne – sam uchwyt nie jest jeszcze wartością w polu!

Uchwyt gwarantuje nam dostęp do wartości całego zestawu atrybutów obiektu, ale koniecznie musimy wskazać, o którą właściwość obiektu nam chodzi – w powyższym przykładzie chcieliśmy odczytać atrybut `value`, czyli wartość typu tekstowego wprowadzoną w polu.

Podstawowe operatory w języku JavaScript

Do podstawowych operatorów, czyli symboli reprezentujących działanie, dostępnych w języku JavaScript zaliczamy:

Operator	Działanie operatora
`+`	dodawanie liczb oraz łączenie łańcuchów (konkatenacja)
`-`	odejmowanie liczb
`*`	mnożenie liczb
`/`	dzielenie liczb
`++`	inkrementacja (zwiększenie wartości liczby dokładnie o 1)
`--`	dekrementacja (zmniejszenie wartości liczby dokładnie o 1)
`%`	reszta z dzielenia (tzw. modulo)
`=`	przypisanie wartości do zmiennej
`==`	porównanie wartości
`<`	mniejsze od
`<=`	mniejsze od lub równe
`>`	większe od
`>=`	większe od lub równe
`!`	zaprzeczenie (negacja)
`&&`	logiczne „i” (`AND`)
`||`	logiczne „lub” (`OR`)
`+=`	skrócony zapis, np. `a += b` odpowiada: `a = a + b`
`-=`	skrócony zapis, np. `a -= b` odpowiada: `a = a - b`

Operator	Działanie operatora
`*=`	skrócony zapis, np. `a *= b` odpowiada: `a = a * b`
`/=`	skrócony zapis, np. `a /= b` odpowiada: `a = a / b`
`%=`	skrócony zapis, np. `a %= b` odpowiada: `a = a % b`

Funkcje i parametry funkcji

Funkcja to **wydzielony, autonomiczny fragment kodu** spełniający określone zadanie. Możemy używać gotowych, wbudowanych w JavaScript funkcji lub tworzyć własne.

Ten zdefiniowany fragment kodu wykona się, jeśli zostanie wywołany do wykonania w konkretnym, zaplanowanym przez programistę miejscu w kodzie, które nazywamy **wywołaniem funkcji**.

Funkcję wywołujemy, zamykając jej nazwę z obu stron okrągłymi nawiasami - dzięki nim wiadomo, że chodzi o funkcję, a nie o np. zmienną:

```
<script>
  // Definicja funkcji
  function iloczyn(a, b)
  {
    return a * b;
    // Zwróć iloczyn wartości parametrów
  }

  var x = 3;
  var y = 4;

  // Wywołanie w wybranym przez nas miejscu w kodzie
  var wynik = iloczyn(x, y);

  // Pokaż wynik wywołania w wyskakującym oknie
  alert(wynik);
</script>
```

Wynikiem działania funkcji dla powyższego wywołania będzie wartość `12`. Zmienne umieszczone wewnątrz nawiasów okrągłych funkcji to jej **parametry**. Analogicznie jak w matematycznym zapisie funkcji `f(x)` parametrem (lub inaczej argumentem) funkcji `f` jest wartość `x`.

Co ważne – wartości: 3 i 4 umieszczone w zmiennych: `x`, `y` występujących w wywołaniu, funkcja na własne potrzeby nazywa: `a`, `b`. Oznacza to, że funkcja **działa na kopiach zmiennych** `x`, `y`.

Rozwiązanie, w którym domyślnie parametr funkcji jest jedynie kopią przekazanej do funkcji oryginalnej wartości, ma wiele zalet. Dzięki temu oryginalne wartości nie ulegną przypadkowej podmianie. W ten sposób zwiększamy bezpieczeństwo danych i oszczędzamy czas, który trzeba by było poświęcić na szukanie ewentualnych skutków zamiany wartości w zmiennych wysyłanych do funkcji. Możemy też łatwiej przenieść własną funkcję do innych źródeł, gdyż jest całkowicie niezależna od nazw zmiennych spoza jej zasięgu.

W języku JavaScript własne funkcje często przypisujemy do obsługi tzw. **zdarzeń** (ang. *event*) zachodzących w przeglądarce – omówmy to teraz bardziej szczegółowo.

Obsługa zdarzeń z użyciem funkcji

Podczas użytkowania internetu łatwo zauważyć, że witryna wyświetlana w przeglądarce internetowej potrafi reagować na działania użytkownika (czyli właśnie zdarzenia) – np. wciśnięcie przycisku, najechanie na element kursorem myszy, wpisanie tekstu w pole edycyjne, wskazanie opcji na liście wyboru etc.

Język JavaScript umożliwia nam stworzenie **obsługi działań użytkownika (zdarzeń)**, czyli przypisanie im wywołania odpowiedniej funkcji. W końcu funkcja to fragment kodu, który wykonuje konkretne zadanie – pasuje to wręcz idealnie do stworzenia różnych reakcji w odpowiedzi na poczynania internauty!

Rozważmy jako przykład obsługę zdarzenia `click` – chcemy, aby po kliknięciu w przycisk z napisem `Hello` skrypt zapytał nas o imię, po czym przywitał się z nami w wyskakującym oknie.

HTML:

```
<input type="button" value="Hello" onclick="powitanie()">
```

JavaScript:

```
<script>
  function powitanie()
  {
    var imie = prompt("Podaj imię");
    alert("Witaj "  + imie);
```

```
  }
</script>
```

Za pomocą atrybutu HTML o nazwie `onclick` przypisujemy do działania użytkownika w przeglądarce wybraną własną funkcję o nazwie `powitanie()`. Poniżej lista kilku najpopularniejszych atrybutów HTML obsługujących zdarzenia w przeglądarce:

Nazwa atrybutu HTML	Zdarzenie, któremu odpowiada atrybut
`onclick`	wyzwalane podczas kliknięcia, kiedy kursor znajduje się na obiekcie
`ondblclick`	wyzwalane podczas dwukrotnego kliknięcia, gdy kursor znajduje się na obiekcie
`onload`	wyzwalane w momencie, gdy cały dokument HTML został załadowany do przeglądarki
`onsubmit`	wyzwalane w momencie przesłania danych z formularza znajdującego się na stronie
`onresize`	wyzwalane w trakcie zmiany rozmiarów obiektu

Co ważne, istnieje także alternatywny, nowy sposób przypisania funkcji do obsługi zdarzenia, polegający na użyciu metody o nazwie `addEventListener()`. Pozwala on dodać do elementu dowolną liczbę tzw. **nasłuchiwaczy zdarzeń**, czyli funkcji, które uruchamiane są w momencie, kiedy dane zdarzenie rzeczywiście zajdzie na stronie.

Przykład z obsługą zdarzenia kliknięcia przycisku zrealizowany w nowej konwencji wygląda następująco:

HTML:

```
<input type="button" value="Hello" id="przycisk">
```

JavaScript:

```
<script>
  function powitanie()
  {
    var imie = prompt("Podaj imię");
    alert("Witaj "  + imie);
  }
```

```
7
8   var przycisk = document.getElementById('przycisk');
9   przycisk.addEventListener("click", powitanie);
10
11 </script>
```

Dzięki tej metodzie podpinania funkcji do obsługi zdarzeń lepiej oddzielamy kod JavaScript od źródła HTML – przypisanie funkcji do zdarzenia nastąpiło tu z użyciem instrukcji języka JavaScript, a nie atrybutu napisanego w HTML. A zatem cały kod JavaScript (w tym przypisanie funkcji do obsługi zdarzeń) możemy przenieść do zewnętrznego pliku z rozszerzeniem `.js`. W języku opisowym HTML definiujemy już tylko identyfikatory obiektów.

Okna dialogowe w przeglądarce

Okna dialogowe (ang. *pop-up window*) często wykorzystywane są w podręcznikowych skryptach oraz podczas egzaminów. Natomiast w codziennej praktyce publikowania witryn w internecie warto zachować umiar w stosowaniu tego typu rozwiązań – strona wyświetlająca wiele wyskakujących informacji będzie irytować użytkownika.

Okno dialogowe `alert`

Jest to informacja dla internauty wyposażona jedynie w przycisk `OK` – posłużyć może do szybkiego wyprowadzenia danych na ekran (instrukcja wyjścia):

```
1 <script>
2   alert("To ja, wyskakujące okienko!");
3 </script>
```

Okno dialogowe `confirm`

Nazwa pochodzi z język angielskiego, gdzie słowo *confirm* oznacza potwierdzenie. Jest to okno dialogowe wyposażone w dwa przyciski: *OK* oraz *Anuluj*, co w kombinacji z instrukcją warunkową `if` może pozwolić nam zmienić zachowanie skryptu w zależności od decyzji użytkownika:

```
1 <script>
2   if(confirm("Podejmij decyzję!")) {
3     alert("Kliknięto: OK");
4    } else {
```

```
    alert("Kliknięto: Anuluj");
 }
</script>
```

Okno dialogowe `prompt`

Umożliwia **wprowadzenie danych** (instrukcja wejścia). W praktyce lepiej jest do tego celu użyć pól edycyjnych i formularza, ale istnieje możliwość pobrania wartości także w oknie dialogowym:

```
<script>
  var imie = prompt("Podaj imię", "Adam");
  alert(imie);
</script>
```

Drugi argument funkcji `prompt()` – w powyższym przykładzie jest to słowo `"Adam"` - to tzw. *placeholder*, czyli wartość domyślna. Będzie ona od razu zaznaczona, tak aby rozpoczęcie pisania usunęło ją z okna dialogowego.

Wszystkie wspomniane wyżej okna dialogowe zaczerpnięte są z interfejsu graficznego systemu operacyjnego i postrzega się je jako przestarzałe. W praktyce warto używać kontrolek formularzy, gdyż są one lepiej odbierane przez użytkowników.

> **Ciekawostka**
>
> Zła „reputacja" okien dialogowych jest skutkiem nadużywania ich w przeszłości w serwisach reklamowych. Obecnie twórcy przeglądarek internetowych wprowadzili skuteczne mechanizmy blokowania zbyt wielu okienek dialogowych wyświetlanych przez jedną witrynę.

Wyprowadzenie wartości zmiennej na ekran

W języku JavaScript istnieje wiele metod wyprowadzenia wartości przechowywanej w zmiennej (czyli tak na prawdę w pamięci RAM) na **płótno przeglądarki** – poznajmy teraz cztery podstawowe sposoby.

Metoda `document.write()`

Największą wadą tej prostej instrukcji jest zniszczenie dotychczasowej zawartości witryny – cały dokument zostanie nadpisany i wypełniony nową, podaną w nawiasie zawartością (na ekranie pozostanie tylko wypisana wartość zmiennej).

```
<script>
  var liczba = 12;
  document.write(liczba);
</script>
```

Zastosowanie okna typu `alert()`

Wartość przechowywaną w zmiennej możemy wyprowadzić w poznanym już oknie dialogowym.

```
<script>
  var liczba = 12;
  alert(liczba);
</script>
```

Użycie elementu struktury HTML

Dzięki znacznikom np. `<div></div>`, `<span></span>` lub `<p></p>` definiujemy dający się uchwycić w języku JavaScript element witryny. Nadajemy mu także **unikalną wartość identyfikatora `id`**, aby można było stworzyć uchwyt tego obiektu.

HTML – przygotowany pojemnik z nadanym identyfikatorem:

```
<div id="wynik"></div>
```

JavaScript – zmiana wartości atrybutu `innerHTML` uchwyconego elementu:

```
<script>
  var liczba = 12;
  document.getElementById("wynik").innerHTML = liczba;
</script>
```

Wyprowadzenie wartości zmiennej nastąpiło dzięki zmianie wartości atrybutu o nazwie `innerHTML` danego pojemnika na stronie. Wartość została wyprowadzona do uprzednio przygotowanego miejsca w witrynie bez niszczenia całego dokumentu, tak jak to miało miejsce w metodzie `document.write()`.

Log konsoli w przeglądarce

Możemy także skorzystać z **lokalnego logu przeglądarki** dostępnego w zakładce `Console`. Aby zobaczyć jego zawartość, należy wywołać `Narzędzia developerskie` – w większości przeglądarek wystarczy w tym celu użyć w witrynie prawego przycisku myszy, po czym wybrać z menu kontekstowego opcję `Zbadaj`.

```
<script>
  var liczba = 12;
  console.log(liczba);
</script>
```

Zdecydowana większość użytkowników nigdy nie zajrzy do lokalnego logu witryny w przeglądarce - jest to narzędzie używane przez programistów, nie zaś klasyczny sposób wyprowadzania na ekran wyników ważnych dla internauty.

Podręczny log witryny warto znać jako alternatywę dla funkcji `alert()` – zwróćmy uwagę, że wyskakujące w przeglądarce okno dialogowe zatrzymuje dalsze wykonanie kodu do momentu, gdy klikniemy w przycisk z napisem *OK*. Log konsoli jest pozbawiony tej uciążliwości, przez co lepiej nadaje się do testowego wyprowadzania wartości nawet wielu zmiennych.

Słownik

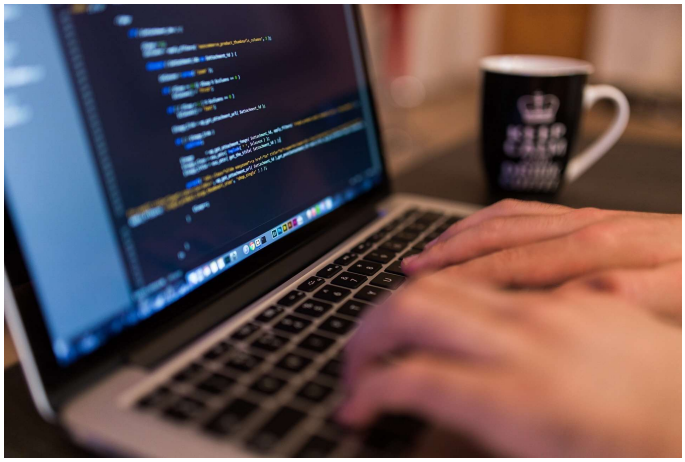
> **hoisting zmiennej**
>
> mechanizm tzw. windowania zasięgu zmiennych w języku JavaScript; zmienna stworzona z użyciem klauzuli `var`, np. wewnątrz funkcji, będzie widoczna nie od momentu jej deklaracji, lecz od samego początku istnienia funkcji, co prowadzi do niepożądanej sytuacji, w której zmiennej można użyć jeszcze przed jej deklaracją
> **front-end**
>
> ogół technologii webowych (HTML, CSS oraz w wielu zastosowaniach JavaScript), w których kody źródłowe wykonywane są przez przeglądarkę internetową klienta witryny oraz pozostają jawne, czyli może do nich zajrzeć każdy użytkownik serwisu, niezależnie od posiadanych uprawnień
> **parsowanie**
>
> proces konwersji tekstowych danych wejściowych pobranych z pliku źródłowego, na odpowiadającą mu wyjściową strukturę obiektową; w procesie tym siłą rzeczy sprawdzana jest poprawność składniowa kodu, czyli zgodność z gramatyką danego języka programowania

Prezentacja multimedialna

Polecenie 1

Zapoznaj się z prezentacją przedstawiającą najpopularniejsze współczesne zastosowania języka JavaScript.

1



Źródło: Andrew Neel, CC 0

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1CYMZQ6P

Bardzo wiele współczesnych technologii webowych wykorzystuje do poprawnego realizowania swoich zadań język JavaScript. Omówmy współczesne zastosowania tego języka. Listę, którą przedstawimy, można także potraktować jako zarys rozbudowanej ścieżki rozwoju programisty JavaScript.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1CYMZQ6P

Klasyczny JavaScript to pełnoprawny, skryptowy język programowania