

Moduł szkoleniowy JavaScript

poziom podstawowy 15 godzinny

Spis treści

1	Metryka dokumentu.....	4
2	Cel.....	5
3	Opis sposobu realizacji celów.....	5
4	Treści kształcenia.....	5
5	Opis założonych osiągnięć ucznia.....	5
6	Sposoby osiągania celów.....	5
6.1	Projekt zaliczeniowy.....	5
7	Propozycje kryteriów oceny i metod sprawdzania osiągnięć ucznia	6
8	Test końcowy sprawdzający wiedzę.....	6
9	Lekcje.....	8
9.1	Lekcja 1 - Wprowadzenie	8
9.1.1	Cel lekcji.....	8
9.1.2	Treść - slajdy z opisem	8
9.1.3	Ćwiczenia	17
9.1.4	Opis założonych osiągnięć ucznia.....	17
9.2	Lekcja 2 – Zmienne, typy danych i operatory.....	17
9.2.1	Cel lekcji.....	17
9.2.2	Treść - slajdy z opisem	18
9.2.3	Ćwiczenia	27
9.2.4	Opis założonych osiągnięć ucznia.....	27
9.3	Lekcja 3 – Instrukcje warunkowe i pętle	27
9.3.1	Cel lekcji.....	27
9.3.2	Treść – slajdy z opisem	27
9.3.3	Ćwiczenia	35
9.3.4	Opis założonych osiągnięć ucznia.....	35
9.4	Lekcja 4 – Tablice.....	36
9.4.1	Cel lekcji.....	36
9.4.2	Treść – slajdy z opisem	36
9.4.3	Ćwiczenia	46
9.4.4	Opis założonych osiągnięć ucznia.....	46
9.5	Lekcja 5 – Obiekty i funkcje	46

9.5.1	Cel lekcji.....	46
9.5.2	Treść – slajdy z opisem	47
9.5.3	Ćwiczenia.....	61
9.5.4	Opis założonych osiągnięć ucznia	61
9.6	Lekcja 6 – Model dokumentu – Document Object Model	61
9.6.1	Cel lekcji.....	61
9.6.2	Treść – slajdy z opisem	62
9.6.3	Ćwiczenia.....	68
9.6.4	Opis założonych osiągnięć ucznia	69
9.7	Lekcja 7 – Zdarzenia	69
9.7.1	Cel lekcji.....	69
9.7.2	Treść – slajdy z opisem	69
9.7.3	Ćwiczenia.....	75
9.7.4	Opis założonych osiągnięć ucznia	75
9.8	Lekcja 8 – Obsługa błędów	75
9.8.1	Cel lekcji.....	75
9.8.2	Treść – slajdy z opisem	75
9.8.3	Ćwiczenie.....	81
9.8.4	Opis założonych osiągnięć ucznia	81
9.9	Lekcja 9 – Wykorzystanie języka JavaScript	81
9.9.1	Cel lekcji.....	81
9.9.2	Treść – slajdy z opisem	82
9.9.3	Ćwiczenia.....	96
9.9.4	Opis założonych osiągnięć ucznia	96
9.10	Lekcja 10 – Biblioteka JQuery	96
9.10.1	Cel lekcji.....	96
9.10.2	Treść – slajdy z opisem	97
9.10.3	Ćwiczenia.....	109
9.10.4	Opis założonych osiągnięć ucznia	109

1 Metryka dokumentu

Szczeciński Park Naukowo Technologiczny			
Dokument	Moduł Szkoleniowy JavaScript		
Krótki opis dokumentu	Moduł szkoleniowy kursu JavaScript zawierający opis celów, treści poszczególnych lekcji, ćwiczeń oraz test końcowy sprawdzający wiedzę		
Data druku	12.02.2013	Liczba stron	109
Nazwa pliku	JavaScript Moduł szkoleniowy.docx	Status	roboczy

Historia zmian

Nr wersji	Data	Opis	Działanie (*)	Autorzy
0.1	01.02.2013	Utworzenie nowego dokumentu	N	Tomasz Czajkowski, Daily Group Sp. z o.o.
1.0	12.02.2013	Wersja gotowa do publikacji	Z	Tomasz Czajkowski, Daily Group Sp. z o.o.

(*) Działanie: N-Nowy, Z-Zmiana, W-Weryfikacja

Lista dystrybucyjna

Imię i nazwisko / Rola	Organizacja

Zgłoszono do odbioru

Imię i nazwisko		Data:		Podpis:	
Imię i nazwisko	Tomasz Czajkowski	Data:	12.02.2013	Podpis:	

2 Cel

Celem kursu jest poznanie na poziomie podstawowym języka programowania obiektowego jakim jest JavaScript, jego składni, podstawowych poleceń i struktur danych. Dodatkowo w trakcie kursu uczestnik pozna kilka praktycznych zastosowań języka przy tworzeniu nowoczesnych, interaktywnych witryn WWW. Zostanie przedstawiona również biblioteka jQuery jako jedno z najpopularniejszych obecnie rozszerzeń języka

3 Opis sposobu realizacji celów

10 półtoragodzinnych lekcji składających się z przypomnienia wiedzy z poprzedniej lekcji, przedstawienia materiału wraz z przykładami, ćwiczeń praktycznych. Po zakończeniu całego cyklu - przeprowadzenie egzaminu sprawdzającego wiedzę.

4 Treści kształcenia

Treść kursu została podzielona na 10 bloków tematycznych – po jednym do każdej lekcji:

1. Wprowadzenie do kursu, podłączenie skryptów do strony www, podstawowe polecenia
2. Zmienne, typy danych i operatory
3. Instrukcje warunkowe i pętle
4. Tablice
5. Obiekty i funkcje
6. Model dokumentu – Document Object Model
7. Zdarzenia
8. Obsługa błędów
9. Wykorzystanie JavaScript
10. Framework jQuery

5 Opis założonych osiągnięć ucznia

Uczestnik po odbyciu kursu pozna podstawy języka JavaScript, pozna składnię i typy danych tego języka. Nauczy się wykorzystywać JavaScript w tworzeniu witryn WWW.

6 Sposoby osiągnięcia celów

Po odbyciu kursu uczeń powinien wykonać projekt/zadanie, które zmotywuje go do pracy indywidualnej ze środowiskiem programistycznym i bazą danych. Zadanie utrwali zdobytą na kursie wiedzę i zmusi do wykorzystania zdobytej wiedzy teoretycznej w praktyce. Zadanie powinno być sformułowane w sposób otwarty, tak aby każdy z uczniów mógł wybrać coś dla niego interesującego. Powinna też zostać dostarczona lista możliwych tematów projektu do wyboru - dla osób, które nie będą miały pomysłu na projekt.

6.1 Projekt zaliczeniowy

1. Utwórz skrypty JS które przygotują animowane menu na witrynie www.

7 Propozycje kryteriów oceny i metod sprawdzania osiągnięć ucznia

Oceną zaliczającą kurs będzie wynik testu sprawdzającego wiedzę z zakresu kursu oraz przygotowanie prostego projektu (dla chętnych).

Kryteria oceny testu:

- 0-7 poprawnych odpowiedzi - 1
- 8-9 poprawnych odpowiedzi - 2
- 10-11 poprawnych odpowiedzi - 3
- 12-13 poprawnych odpowiedzi - 4
- 14-15 poprawnych odpowiedzi - 5

8 Test końcowy sprawdzający wiedzę

Krótki (ok 20 minut) test składający się z 15 pytań

Przykładowy test (pogrubioną czcionką zaznaczono poprawne odpowiedzi)

1. Wewnątrz jakiego elementu należy umieścić treść skryptu
 - a. `<js>`
 - b. `<script>`**
 - c. `<javascript>`
 - d. `<scripting>`
2. Jaka jest prawidłowa składnia polecenia wypisującego tekst „Witaj świecie!”
 - a. `("Witaj świecie!");`
 - b. `"Witaj świecie!";`
 - c. `response.write("Witaj świecie!");`
 - d. `document.write("Witaj świecie!");`**
3. Jaka jest poprawna składnia polecenia ładującego zewnętrzny plik skrypt.js?
 - a. `<script type="text/javascript" name="skrypt.js">`
 - b. `include("skrypt.js");`
 - c. `<script type="text/javascript" src="skrypt.js">`**
 - d. `<script type="text/javascript" href="skrypt.js">`
4. W jaki sposób tworzymy funkcję w JavaScript
 - a. `function = mojaFunkcja()`
 - b. `function:mojaFunkcja()`
 - c. `function mojaFunkcja()`**
5. Jaki jest poprawny sposób utworzenia tablicy JavaScript?
 - a. `var tab = new Array("Jan", "Adam", "Jacek");`**
 - b. `var tab = new Array(1:"Jan", 2:"Adam", 3:"Jacek");`
 - c. `var tab = new Array:1=("Jan")2=("Adam")3=("Jacek");`

- d. `var tab = new Array = "Jan", "Adam", "Jacek"`
6. W jaki sposób można wywołać funkcję o nazwie `mojaFunkcja`?
- `exec mojaFunkcja()`
 - `mojaFunkcja()`**
 - `call mojaFunkcja()`
 - `call function mojaFunkcja()`
7. W jaki sposób zapisać instrukcję warunkową jeśli `i` jest równe 5 to wykonaj instrukcje
- `if i=5 then instrukcje;`
 - `check(i = 5) instrukcje`
 - `if (i == 5) { instrukcje }`**
 - `if (i == 5) then { instrukcje }`
8. Jak definiujemy pętlę While
- `while (i <= 10)`**
 - `while i= 1 to 10`
 - `while(i <= 10; i++)`
9. Warunek pętli do ... while jest sprawdzany
- przed każdą iteracją pętli
 - po każdej iteracji pętli**
 - raz – na końcu pętli
 - raz – na początku pętli
10. Jakie zdarzenie sprawdza czy pole formularza zmieniło wartość
- `onsubmit`
 - `onblur`
 - `onchange`**
 - `onclick`
11. Ile parametrów można przekazać do funkcji
- żadnego
 - tyle ile chcesz
 - jeden na każdy argument funkcji**
 - jeden
12. Kiedy zdarzenie JavaScript nie zostanie uruchomione
- kiedy inne zdarzenie jest jeszcze obsługiwane
 - kiedy JavaScript jest wyłączony w przeglądarce**
 - kiedy strona używa arkuszy stylu CSS
 - kiedy strona uruchamiana jest lokalnie a nie na serwerze
13. Tablica w JavaScript jest:
- zmienną
 - obiektem**
 - metodą
 - funkcją
14. W jaki sposób można zmienić tekst zawierający przecinki na tablicę
- `tablica = tekst.indexOf(",")`
 - `tablica = tekst.split(",");`**

- c. `tablica = tekst.trim(",");`
 - d. `tablica = tekst.substring(",");`
15. W którym obiekcie DOM dostępna jest treść strony
- a. `window`
 - b. `browser`
 - c. **`document`**
 - d. `location`

9 Lekcje

9.1 Lekcja 1 - Wprowadzenie

9.1.1 Cel lekcji

Celem lekcji jest wprowadzenie do tematyki związanej z programowaniem interaktywnych stron internetowych. Uczestnik pozna podstawowe informacje o tworzeniu stron WWW z wykorzystaniem JavaScript, historię tego języka oraz sposoby umieszczania skryptów w plikach html.

9.1.2 Treść - slajdy z opisem

Slajd 1



W kursie przedstawimy podstawy języka JavaScript, jego podstawowe polecenia, typy danych. Zaprezentujemy sposoby jego wykorzystania w witrynach WWW. Przedstawimy bibliotekę jQuery jako obecnie najpopularniejszą i wykorzystywaną na większości stron WWW.

Zacniemy od informacji co to jest JavaScript, w jaki sposób można go podłączyć do strony www i do czego można go wykorzystać. W kolejnych lekcjach wprowadzimy podstawowe pojęcia i instrukcje języka JavaScript, pokażemy model obiektowy języka, model dokumentu html i możliwości reagowania na działania użytkownika.



Slajd 2

JavaScript

- język skryptowy
- nie wymaga kompilowania przed uruchomieniem
- wykonywany przez program zwany interpreterem/parserem
- parser jest wbudowany we większość przeglądarek

JavaScript to język tekstowy który nie wymaga żadnej konwersji przed uruchomieniem. Inne języki takie jak Java i C++ muszą zostać skompilowane zanim będzie można je uruchomić, natomiast JavaScript jest uruchamiany od razu przez program zwany parserem lub interpreterem. Praktycznie każda nowoczesna przeglądarka zawiera w sobie parser języka JavaScript.

JavaScript nie jest używany jako samodzielny język, został bowiem zaprojektowany do łatwego osadzania w innych produktach i aplikacjach, jak na przykład przeglądarkach internetowych. Wewnątrz swojego bazowego środowiska, JavaScript może być połączony z obiektami otoczenia, w którym się znajduje, aby zapewnić nad nim programową kontrolę.

Wykorzystywany w witrynach www do zapewnienia interakcji z użytkownikiem.

Slajd 3

Krótką historia

- stworzony przez firmę Netscape w 1995 roku.
- Głównym twórcą jest Brandon Eich
- początkowo nazywany "Mocha", później "LiveScript"
- po podpisaniu umowy z SUN Microsystems wprowadzono nazwę JavaScript
- Organizacja ECMA rozpoczęła pracę nad standardem języka w 1996 roku, standard został nazwany ECMAScript
- obecna wersja to 5.1 z czerwca 2011

W roku 1995 Netscape Navigator był dominującą przeglądarką na rynku i firma zdecydowała dodać interaktywność do stron HTML dzięki lekkiemu językowi programowania. Prace nad językiem zostały zlecone Brandanowi Eichowi, który napisał pierwszą wersję języka w 10 dni. Wstępna nazwa "Mocha" została zmieniona na "LiveScript" a w wyniku porozumienia z firmą Sun Microsystems na "JavaScript".

Język ten miał służyć jako klej łączący aplety wykonane w technologii Java, dodatkowo chciano wykorzystać szum medialny związany z Javą.



Pod taką nazwą został opublikowany w przeglądarce Netscape Navigator 2.0B3.

W 1996 roku rozpoczęły się prace nad opracowaniem standardu przez organizację standaryzującą ECMA International. Z racji tego, że Sun był właścicielem marki Java nowy standard nie mógł zostać nazwany

JavaScript. W wyniku tego nazwa standardu to ECMAScript a jego implementacje to JavaScript, JScript (Microsoft) itd.


Obecna wersja ECMAScript to 5.1 – jest ona implementowana przez większość nowych przeglądarek (Chrome w wersji 19+, Mozilla Firefox 4+, Internet Explorer 9+)

Slajd 4

Dodawanie JS do strony www



- `<script type="text/javascript" language="JavaScript 1.5">`
- `alert(12 > 6);`
- `</script>`



W języku HTML za umieszczanie skryptów JS odpowiedzialny jest element `<script>` z argumentem `type` o wartości `text/javascript`, `application/javascript` oraz argumentem `language` o wartości `javascript`.


Atrybut `language` jest jednak przestarzały i zaleca się go pomijać. Jest on pozostałością z czasów, kiedy istniał jeszcze język skryptowy VBScript żeby odróżnić skrypty napisane w tych językach

Slajd 5

Zgodność ze starszymi przeglądarkami

- `<script type="text/javascript" language="JavaScript 1.5">`
- `<!--`
- `alert(12 > 6);`
- `//-->`
- `</script>`



Przeglądarki nie obsługujące skryptów nie potrafią zinterpretować znacznika `<script>`. Podstawowym działaniem przeglądarki jest ignorowanie znaczników, których nie rozumieją. Takie działanie jest poprawne, gdy znacznik jest pojedynczym wyrażeniem, ale w znaczniku `<script></script>` można umieścić wiele wyrażeń.



Starsze przeglądarki nie wiedzą, że mają się spodziewać znacznika zamykającego `</script>` więc po prostu wygenerują wszystkie linie występujące po otwarciu znacznika `<script>`.

Aby rozwiązać ten problem można otoczyć wyrażenia javascript znacznikami komentarza html `<!-- -->`. linia zamykająca komentarz html

rozpoczyna się znakiem komentarza JS (//), który oznacza polecenie zignorowania tej linii przez interpreter JavaScript, ale przeglądarki nie wspierające skryptów zinterpretują znacznik końca komentarza i rozpoczną generowanie strony od kolejnego znacznika.

Rozwiązanie to zapewniało zgodność ze starszymi przeglądarkami (obecnie bardzo starymi :) Obecnie można ten mechanizm pominąć, ale przytoczymy go tutaj dla porządku

Slajd 6

 **B2E**
BUSINESS TO EDUCATION  **SZCZECIŃSKI PARK**
NAUKOWO-TECHNOLOGICZNY

Zgodność z XHTML Strict

- `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"`
`"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
- `<script type="text/javascript">`
- `/* */</code>• <code>var x = 3;</code>• <code>alert("Witajcie! - x is ' + x);</code>• <code>/*]]> */</code>• <code></script></code></div><div data-bbox="557 346 884 406" data-label="Text"><p>Jeśli jako typ dokumentu zdefiniujemy Strict XHTML (<code><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"</code></p></div><div data-bbox="557 405 884 452" data-label="Text"><p><code>"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"></code>) musimy otoczyć skrypt znacznikiem CDATA.</p></div><div data-bbox="557 466 884 586" data-label="Text"><p>Parsery XHTML nie pozwalają na symbole '<code><</code>' i '<code>></code>', które nie są elementami xml, więc treść skryptu należy umieścić w znaczniku specjalnym [CDATA], który nakazuje interpreterowi XML traktować wyrażenia wewnątrz znacznika jako treść.</p></div><div data-bbox="557 586 884 678" data-label="Text"><p>W XHTML nie można używać argumentu language w znaczeniu określenia wersji języka JS (atrybut, jeżeli jest użyty, powinien przyjąć dwuznakowe wartości opisane standardem ISO 639, np. EN, DE, PL)</p></div><div data-bbox="599 893 882 908" data-label="Page-Footer"><p>Moduł szkoleniowy JavaScript str. 11</p></div><div data-bbox="410 913 586 927" data-label="Page-Footer"><p>Człowiek - najlepsza inwestycja</p></div><div data-bbox="212 933 337 962" data-label="Page-Footer"> KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI</div><div data-bbox="649 936 776 959" data-label="Page-Footer"> UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY</div><div data-bbox="228 965 766 978" data-label="Page-Footer"><p>Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego</p></div>`



Slajd 7

Podłączenie zewnętrznego pliku

- Zalecany sposób:
- `<script type="text/javascript" src="plik.js"></script>`

Zaleca się jednak przechowywać skrypty w osobnych plikach i podłączać je do stron za pomocą atrybutu src

```
<script  
type="text/javascript" src="plik.js"></sc  
ript>
```

Plik "plik.js" zawiera tylko definicję skryptów (bez dodatkowych znaczników <script>). Ogólnie przyjętą praktyką jest aby plik miał rozszerzenie ".js", chociaż nie jest to wymagane.

Zaletą takiego podejścia jest możliwość ponownego użycia skryptu na innej stronie.

Jeśli skrypt będzie zagnieżdżony w znaczniku HTML, każda strona będzie zawierała nadmiarową treść i modyfikacja skryptu będzie wymagała takich samych zmian w każdym pliku.

Z kolei podłączenie zewnętrznego skryptu to tylko jedna dodatkowa linia w każdym pliku html i zewnętrzny skrypt może być zmodyfikowany raz, aby zmiany były widoczne na wszystkich stronach, które go wykorzystują.

Plik ze skryptem zostanie umieszczony w pamięci podręcznej przeglądarki (cache) i wykorzystywany przy przechodzeniu między witrynami na stronie.

Dodatkową zaletą jest brak konieczności stosowania komentarzy i znaczników CDATA.



Slajd 8

Znacznik noscript

- `<noscript>`
- `<p>`
- Twoja przeglądarka nie obsługuje JavaScriptu. Aby zobaczyć stronę w pełnej funkcjonalności, zainstaluj inną przeglądarkę lub włącz opcję wyświetlania JS w przeglądarce
- `</p>`
- `</noscript>`

Dodatkowym znacznikiem który możemy wykorzystać jest znacznik `<noscript></noscript>`. W przypadku nowoczesnych przeglądarek z wyłączoną obsługą JavaScript zostanie wyświetlona treść znajdująca się między znacznikami `<noscript>`. Możemy umieścić tam informację o konieczności włączenia obsługi skryptów, lub prośbę o wyświetlenie naszej strony w innej przeglądarce. Starsze przeglądarki również wyświetlą taką informację (wynika to z wcześniejszego wspomnianego mechanizmu działania przeglądarek - nieznany znacznik jest pomijany, a zawartość jest traktowana jako treść do wygenerowania i wysyłana do przeglądarki).

Slajd 9

Gdzie umieszczać skrypty?

- w nagłówku strony
- `<!DOCTYPE html...>`
- `<html>`
- `<head>`
- `<title>Tytuł</title>`
- `<script src="plik.js"></script>`
- `</head>`
- `<body>`
- `<!-- treść strony -->`
- `</body>`
- `</html>`

Technicznie można umieścić znacznik `<script>` w dowolnym miejscu na stronie. Może to być sekcja `<head>` lub sekcja `<body>`. Skrypty znajdujące się w nagłówku strony zostaną wykonane jeszcze przed załadowaniem właściwej treści strony, natomiast w przypadku umieszczenia skryptu w ciele strony przed uruchomieniem skryptu zostanie wygenerowana część strony znajdująca się przed znacznikiem `<script>`

Klasyczne najlepsze praktyki wskazują umieszczenie skryptów w nagłówku strony. Zaletą takiego podejścia jest jak wspomnieliśmy wczytanie całego skryptu przed załadowaniem strony. Dodatkowo łatwiej jest innym programistom znaleźć znacznik `script`, co ułatwia debugowanie strony.

Natomiast wady takiego rozwiązania są następujące:

- ładowanie treści jest wstrzymane aż do momentu aż załadują i wykonają się wszystkie skrypty,
- skrypty nie mają dostępu do znaczników HTML w dokumencie, ponieważ nie został on jeszcze wczytany. Trzeba opóźnić

wywołanie skryptów modyfikujących stronę, aż do momentu jej pełnego załadowania (można to zrobić za pomocą zdarzeń, które omówimy w późniejszych lekcjach).

Slajd 10

Gdzie umieszczać skrypty?

- na końcu sekcji <body>

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Tytuł</title>
</head>
<body>
  <!-- treść strony -->
  <script src="myscripts.js"></script>
</body>
</html>
  
```



Niektórzy specjaliści od wydajności zalecają umieszczanie skryptów na końcu sekcji <body>. Ładowanie strony nie jest przerywane przez wczytywanie skryptów, w skrypcie nie trzeba czekać na zakończenie ładowania strony ponieważ cała treść jest już dostępna.

Wadą są takie same :) strona jest dostępna dla użytkownika jeszcze zanim zostaną wgrane pliki JavaScript. Oznacza to, że jeśli np. na naszej witrynie zastosowaliśmy formularz, którego poprawne wypełnienie sprawdzamy za pomocą skryptu, to użytkownik może wysłać taki formularz zanim skrypt sprawdzający będzie dostępny.

Aby rozwiązać taki problem możemy podzielić nasze skrypty na dwie grupy – jedną zawierającą ważne funkcje dołączyć do strony w nagłówku, a drugą z mniej ważnymi funkcjami w stopce strony.




Slajd 11



Bezpieczeństwo JavaScript

- JavaScript nie zawiera żadnych mechanizmów bezpieczeństwa.
- Każdy skrypt ma takie same prawa – każdy może uzyskać dostęp do innego skryptu i zmodyfikować jego zmienne i funkcje.
- Skrypt może nadpisać nawet natywne funkcje JavaScriptu.





JavaScript nie zawiera żadnych mechanizmów bezpieczeństwa.

Każdy skrypt ma takie same prawa – każdy może uzyskać dostęp do innego skryptu i zmodyfikować jego zmienne i funkcje.

Skrypty mogą odczytywać pliki cookies, i wykorzystując prototyp funkcji można nadpisać zawartość każdej natywnej funkcji JavaScript.


Dodatkowo JavaScript można wyłączyć, więc nie należy blokować dostępu do strony za pomocą mechanizmów JavaScript.

Slajd 12



Dlaczego używać JavaScriptu?



- uruchamiany po stronie klienta
- ogranicza czas wykonywania strony (strona nie jest wysyłana na serwer i odsyłana do klienta po przetworzeniu)



W czasach gdy ceny hostingu były stosunkowo wysokie, a prędkości połączenia z internetem niskie możliwość wykonywania kodu po stronie klienta była nieoceniona. Można było reagować na działania użytkowników bez konieczności przesyłania strony do serwera. Strony z dobrze wykorzystanymi możliwościami JavaScript były bardziej interaktywne i szybsze.


Obecnie prędkości i ceny są inne, ale brak konieczności odsyłania stron na serwer nadal powoduje że interakcja z nimi jest dużo szybsza i płynniejsza.

Slajd 13



Do czego można wykorzystać JS

- JavaScript jest prosty w implementacji
- działa na komputerze użytkownika
- dodaje dynamiczną zawartość na stronie (menu itp.)
- może załadować treść do strony bez przesyłania jej na serwer (AJAX)
- w skrypcie można sprawdzić, czy dana funkcjonalność jest dostępna w przeglądarce i odpowiednio zareagować,
- javascript może być wykorzystany w celu poprawy błędów np. obsługa stylu css przez przeglądarkę



Do czego można wykorzystać JavaScript?



JavaScript jest prosty w implementacji – wystarczy napisać prosty plik tekstowy i podłączyć go do strony WWW,

działa po stronie klienta – zapewnia szybszą reakcję na działania użytkownika i ogranicza zużycie zasobów po stronie serwera, dodaje dynamiczną zawartość na stronie – nie trzeba przysyłać strony na serwer żeby wykonać jakąś zmianę w treści,

może załadować treść w momencie gdy użytkownik jej potrzebuje – bez konieczności przeładowania całej


strony (AJAX),
w skrypcie można sprawdzić, czy
dana funkcjonalność jest dostępna w
przeglądarce i odpowiednio
zareagować,
javascript może być wykorzystany w
celu poprawy błędów np. obsługi
stylu css przez przeglądarkę

Slajd 14

 **B2E**
BUSINESS TO EDUCATION
 **SZCZECIŃSKI PARK**
NAUKOWO-TECHNOLOGICZNY

Zastosowania JavaScript

- Dawniej służył głównie do obsługi formularzy
 - alert, confirm, prompt
- Obecnie:
 - dostęp do elementów strony, modyfikacja ich wyglądu, treści i atrybutów
 - tworzenie nowych elementów na stronie
 - podpowiadanie tekstu w wyszukiwarce, a nawet zwracanie wstępnych wyników

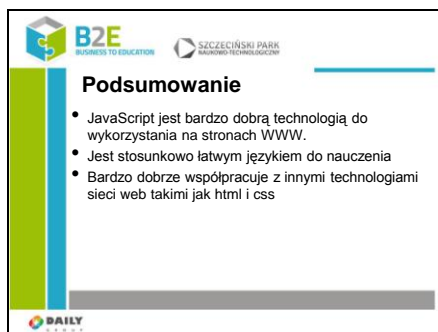


Dawniej służył głównie do obsługi
formularzy
- alert, confirm, prompt

Obecnie dzięki obsłudze DOM można
odczytać dowolny element na stronie i
zmieniać jego wygląd, zawartość i
atrybuty. Możliwe jest także tworzenie
nowych elementów.

Tym samym nie musimy obecnie
wykorzystywać funkcji alert, aby
wyświetlić komunikat użytkownikowi.
Możemy dodać dodatkowy element na
stronie, w którym wpiszemy treść
naszego komunikatu. JavaScript może
zostać również wykorzystany do
tworzenia podpowiedzi w oknach
wyszukiwania, a nawet zwracania
wstępnych wyników jeszcze w trakcie
pisanie zapytania.

Slajd 15



Podsumowanie

- JavaScript jest bardzo dobrą technologią do wykorzystania na stronach WWW.
- Jest stosunkowo łatwym językiem do nauczania
- Bardzo dobrze współpracuje z innymi technologiami sieci web takimi jak html i css

JavaScript jest bardzo dobrą technologią do wykorzystania na stronach WWW.

Jest stosunkowo łatwym językiem do nauczania

Bardzo dobrze współpracuje z innymi technologiami sieci web takimi jak html i css

9.1.3 Ćwiczenia

Utwórz przykładową stronę www, zawierającą podstawowe znaczniki html i javascript. Utworzoną stronę będziemy wykorzystywać jako przykłady w innych lekcjach

9.1.4 Opis założonych osiągnięć ucznia

Po tej lekcji uczestnicy będą znać podstawowe zastosowania języka JavaScript związane z obsługą stron WWW, będą potrafili utworzyć prostą stronę internetową wykorzystującą JavaScript

9.2 Lekcja 2 – Zmienne, typy danych i operatory

9.2.1 Cel lekcji

Celem lekcji jest zapoznanie uczestników z podstawowymi typami danych dostępnymi w języku JavaScript, sposobami definiowania zmiennych oraz z podstawowymi operatorami służącymi do manipulacji wartością zmiennych skryptu.

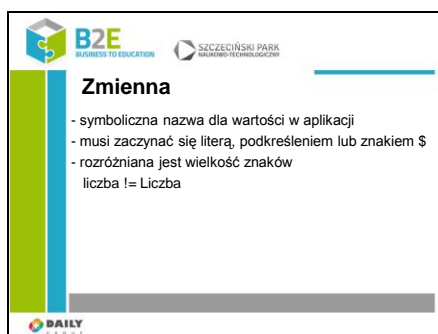
9.2.2 Treść - slajdy z opisem

Slajd 1



W poprzedniej lekcji wspomnieliśmy o zastosowaniach języka JavaScript i sposobach podłączania skryptów js w naszej witrynie. W tej lekcji zajmiemy się omówieniem podstawowych elementów służących do przechowywania danych oraz zobaczymy jakie podstawowe typy danych są dla nas dostępne. Zaczniemy od wprowadzenia pojęcia zmiennej.

Slajd 2





Zmiennych używa się jako symbolicznych nazw dla wartości w Twojej aplikacji. Każda zmienna ma swoją nazwę, która pozwala na identyfikację tej zmiennej w kodzie skryptu. Zmienna charakteryzuje się również typem, który określa rodzaj danych jakie przechowuje.

Nazwy zmiennych, nazywane *identyfikatorami*, podporządkowane są pewnym regułom.

Identyfikator JavaScript musi zaczynać się literą, podkreśleniem (`_`) lub znakiem dolara (`$`); kolejne znaki mogą być cyframi (0-9). Ponieważ JavaScript rozróżnia duże/małe litery, litery oznaczają znaki od "A" do "Z" (wielkie litery) oraz znaki od "a" do "z" (małe litery).




Slajd 3



Deklaracja zmiennych



Zmienną można zadeklarować na 2 sposoby:

- za pomocą słowa kluczowego var. Na przykład var liczba = 10, var nazwisko = "Kowalski"
- za pomocą prostego przypisania wartości. Na przykład liczba = 10, nazwisko = "Kowalski"




zmienna zadeklarowana za pomocą słowa kluczowego var jest zmienną lokalną, natomiast bez słowa kluczowego zmienną globalną (więcej informacji na temat zmiennych lokalnych i globalnych będzie w późniejszych lekcjach)

Slajd 4



Typy danych

- typ liczbowy
- typ łańcuchowy
- typ logiczny
- typ specjalny





Typ danych określa rodzaj danych jaki przechowuje.

W JavaScriptcie możemy podzielić typy na następujące rodzaje:

- typ liczbowy
- typ łańcuchowy
- typ logiczny
- typ specjalny


Slajd 5



Typ liczbowy

- służy do reprezentacji liczb.
- liczby zapisywane są za pomocą literałów liczbowych, czyli ciągów znaków składających się na liczbę

```
var pi = 3,1415;  
var licznik = 10;
```





Typ liczbowy służy do reprezentacji liczb, przy czym nie ma występującego w klasycznych językach programowania rozróżnienia na typy całkowitoliczbowe i rzeczywiste (zmiennopozycyjne). Liczby zapisywane są za pomocą literałów (stałych napisowych, z ang. string constant, literal constant) liczbowych, czyli ciągów znaków składających się na liczbę. Obowiązują przy tym następujące zasady.

- Jeżeli ciąg cyfr nie jest poprzedzony żadnym znakiem lub jest poprzedzony znakiem +, reprezentuje wartość dodatnią, jeżeli natomiast jest poprzedzony znakiem -, reprezentuje wartość ujemną.

- Jeżeli literał rozpoczyna się od cyfry 0, jest traktowany jako wartość ósemkowa.
- Jeżeli literał rozpoczyna się od ciągu znaków 0x, jest traktowany jako wartość szesnastkowa (heksadecymalna). W zapisie wartości szesnastkowych mogą być wykorzystywane zarówno małe, jak i wielkie litery alfabetu, od A do F.
- Literały mogą być zapisywane w notacji wykładniczej, w postaci $X.YeZ$, gdzie to część całkowita, Y — część dziesiętna, natomiast Z to wykładnik potęgi liczby 10. Zapis taki oznacza to samo, co $X.Y * 10Z$


Slajd 6

 **B2E**
BUSINESS TO EDUCATION
 **SZCZECIŃSKI PARK**
NAUKOWO-TECHNOLOGICZNY

Typ łańcuchowy

- służy do reprezentacji ciągów znaków (napisów).
- powinni być ujęte w znaki cudzysłowu, aczkolwiek dopuszczalne jest również wykorzystanie znaków apostrofu.

```
var napis = "Witaj świecie!"  
var nazwisko = "Kowalski";
```





Typ łańcuchowy (inaczej, typ string) służy do reprezentacji ciągów znaków (napisów). Ciągi te (inaczej, stałe napisowe) powinny być ujęte w znaki cudzysłowu, aczkolwiek dopuszczalne jest również wykorzystanie znaków apostrofu.

Przykładowy ciąg ma postać:
"abcdefg" lub 'abcdefg'



Slajd 7




Typ łańcuchowy

Ciągi mogą też zawierać poniższe znaki specjalne

- \b - backspace
- \n - nowa linia
- \r - powrót karetki
- \f - nowa strona
- \t - tabulacja
- \" - cudzysłów
- \' - apostrof
- \\ - lewy ukośnik


var cytat = "Marcin powiedział: \"Jedźmy na wycieczkę\"";



Ciągi mogą też zawierać poniższe znaki specjalne


- \b - backspace
- \n - nowa linia
- \r - powrót karetki
- \f - nowa strona
- \t - tabulacja
- \" - cudzysłów
- \' - apostrof
- \\ - lewy ukośnik

Slajd 8





Typ logiczny

- pozwala na określenie dwóch wartości logicznych: prawda i fałsz.
- wartości tego typu są wykorzystywane przy konstruowaniu wyrażeń logicznych, porównywaniu danych itp.



Typ logiczny (boolean) pozwala na określenie dwóch wartości logicznych: prawda i fałsz. Wartość prawda jest w JavaScriptcie reprezentowana przez słowo true, natomiast wartość fałsz przez false. Wartości tego typu są używane przy konstruowaniu wyrażeń logicznych, porównywaniu danych, wskazywaniu, czy dana operacja zakończyła się sukcesem itp.


Slajd 9



Typ specjalny



2 wartości:

- null
- undefined



Możemy wyróżnić dwa rodzaje typów specjalnych: null i undefined. Choć podobne, nie są tożsame. typ null to specjalne słowo oznaczające wartość null (pustą), ponieważ JavaScript rozróżnia małe/duże litery, null to nie to samo co Null, NULL czy jakkolwiek inaczej. typ undefined to podstawowa właściwość, której wartość jest nieokreślona. W tym kontekście wartość undefined ma niezainicjalizowaną zmienną, zmienną, której jawnie przypisano wartość undefined, bądź też nieistniejąca właściwość obiektu.

Slajd 10




Zmienne a typy danych

- zmienne mogą zmieniać swój typ w trakcie działania programu

```
var mojaZmienna = "12345"
```

```
mojaZmienna = mojaZmienna - 345
```

wynik: 12000



W JavaScriptcie w przeciwieństwie do wielu języków programowania zmienna może automatycznie zmieniać swój typ. Np. zmienna łańcuchowa może zostać zamieniona na zmienną liczbową lub odwrotnie w zależności od sposobu odwołania do zmiennej.

Np. poniższa deklaracja przypisze zmiennej `mojaZmienna` wartość typu tekst "12345"

```
var mojaZmienna = "12345"
```

Pomimo tego, że ten łańcuch składa się z samych cyfr jest zmienną tekstową. Jednak JavaScript jest na tyle sprytny, że w poniższym przypisaniu zamieni typ znakowy na liczbę przed odejmowaniem i wynikową wartość (w tym przypadku liczba 12000) zapisze ponownie do zmiennej `mojaZmienna`

```
mojaZmienna = mojaZmienna - 345
```



Slajd 11

Zmienne a typy danych cd.

- Ale:
- `var mojaZmienna = "12345"`
- `mojaZmienna = mojaZmienna + 678`
- Wynik = „12345678”
- `mojaZmienna = Number(mojaZmienna) + 678`

Logos: B2E, SZCZECIŃSKI PARK, DAILY

Niestety taka dynamiczna konwersja ma swoje wady. W przypadku przypisania

```
var mojaZmienna = "12345"
```

```
mojaZmienna = mojaZmienna + 678
```

interpreter JS nie zmieni łańcucha "12345" na liczbę tylko zamieni liczbę 678 na łańcuch "678" i połączy oba łańcuchy co da w wyniku "12345678".

Aby zapobiec takiej sytuacji należy wskazać, że mojaZmienna jest typem liczbowym

```
mojaZmienna = Number(mojaZmienna) + 678
```

Taka instrukcja nazywana jest rzutowaniem na typ. W języku JavaScript dostępne są następujące funkcje do zmiany typu zmiennej:

`Boolean()` - zmienia wartość na prawdę lub fałsz

`Number()` - zmienia wartość na liczbową

`String()` - zmienia wartość na łańcuch znaków

Przy rzutowaniu na wartość logiczną łańcuch znaków o minimalnej długości 1, dowolny obiekt lub liczba większa od 0 będzie miała wartość "true", z kolei pusty łańcuch znaków, liczba 0 oraz typy specjalne null i undefined będą miały wartość "false"

Jeśli wartość rzutowana na typ liczbowy, zawiera cokolwiek oprócz cyfr to wynikiem jest NaN (not a number).

Funkcja `String()` zwraca dowolną wartość jako łańcuch znaków, nawet typy specjalne jak null i undefined.



Slajd 12

Instrukcje

polecenie do wykonania, zakończone średnikiem możemy np.

- wyświetlić okno dialogowe,
- umieścić treść na stronie www,
- zmodyfikować styl warstwy

Instrukcję możemy potraktować jak polecenie do wykonania. Skrypt napisany w języku JavaScript to w istocie ciąg instrukcji, a zatem ciąg poleceń. Możemy zatem nakazać wyświetlenie okna dialogowego, umieścić jakąś treść na stronie WWW, zmodyfikować zawartość warstwy czy innego elementu witryny itp.

Co ważne, każdą instrukcję powinniśmy zakończyć znakiem średnika. To informacja dla interpretera, że jest to koniec instrukcji.

Przykładowe instrukcje:

```
var zmienna = 10;
```

```
var element = document.getElementById("content");  
content.innerHTML = "Tekst";
```

Slajd 13

Przykłady instrukcji

Przykładowe instrukcje:

```
var zmienna = 10;  
var element = document.getElementById("content");  
content.innerHTML = "Tekst";
```


Slajd 14

Operatory

- arytmetyczne,
- porównywania (relacyjne),
- logiczne,
- przypisania,

W JavaScriptcie, podobnie jak w innych językach programowania, występują operatory pozwalające na wykonywanie rozmaitych operacji. Operatory możemy podzielić na następujące grupy:

- arytmetyczne,
- porównywania (relacyjne),
- logiczne,
- przypisania,

Slajd 15

Operatory arytmetyczne

Operator	Opis	Przykład	Wynik
+	Dodawanie	3 + 11	14
-	Odejmowanie	9 - 4	5
*	Mnożenie	3 * 4	12
/	Dzielenie	21 / 7	3
%	Reszta z dzielenia	21 % 8	5
++	Zwiększanie	a= 5; ++a	6
--	Zmniejszanie	a= 5; --a	4

Operatory arytmetyczne w JavaScript pozwalają na tworzenie wyrażeń numerycznych.

Operator % to operator modulo - podaje jako wynik resztę z dzielenia, operatory ++ i -- są uproszczonymi wersjami operatorów dodawania o 1 i odejmowania o 1

Slajd 16

Operatory porównania

Operator	Opis	Przykład	Wynik
===	Równe	1 === 1	Prawda
===	Równe w wartości i typie	1 === '1'	Falsz
!=	Nie równe	1 != 2	Prawda
!=	Nie równe w wartości i typie	1 != '1'	Prawda
>	Większe niż	1 > 2	Falsz
<	Mniejsze niż	1 < 2	Prawda
>=	Większe lub równe	1 >= 1	Prawda
<=	Mniejsze lub równe	2 <= 1	Falsz

Operator == i === różnią się sposobem dokładnością sprawdzania warunek if (1 == '1') da w wyniku wartość "true" ponieważ '1' może być automatycznie zamienione na 1. natomiast warunek if (1 === '1') da w wyniku wartość "false" ponieważ mimo że wartości mogą być uznane za takie same to zmienne są różnego typu

Slajd 17

Operatory logiczne			
Operator	Opis	Przykład	Wynik
&&	I	1 == 1 && 2 == 2	Prawda
	Lub	1 == 1 2 == 3	Prawda
!	Negacja	!(1 == 1)	Fałsz

JavaScript wspiera 3 operatory logiczne, za pomocą których można rozszerzać instrukcję if.

Operatory w języku JavaScript sprawdzane są w kierunku od lewej do prawej. Tzn jeśli w warunku zawierającym operator && na pierwszym miejscu będzie wyrażenie którego wartość nie będzie spełniona to sprawdzanie całego warunku zostanie przerwane i jako wynik będzie zwrócona wartość Fałsz

Slajd 18

Operatory przypisania			
Operator	Opis	Przykład	Wynik w a
=	Proste przypisanie	a = 42	42
+=	Przypisanie z dodawaniem	a += 5	47
-=	Przypisanie z odejmowaniem	a -= 2	45
*=	Przypisanie z mnożeniem	a *= 2	90
/=	Przypisanie z dzieleniem	a /= 10	9
%=	Przypisanie z modulo	a %= 4	1

Przykładowo zamiast pisać $a = a + 5$ można użyć prostszego $a += 5$.

Powyższych operatorów można używać w połączeniu z innymi operatorami i zmiennymi

$a = 10;$

$b = 25;$

$a += (b / 5)$

wynikiem operacji jest 15 ($10 + (25/5)$)

Slajd 19

Kolejność operatorów			
Priorytet	Operatory	Priorytet	Operatory
1	[] new	10	&
2	()	11	^
3	++ --	12	!
4	! ~ unary+ unary- typed delete	13	&&
5	~ /%	14	
6	++ --	15	?:
7	<< >> <<< >>>	16	in this case the % operator has the lowest priority
8	< <= > >= in instanceof	17	,
9	new this case the % operator has the lowest priority		

Oprócz znajomości operatorów, niezbędna jest jeszcze wiedza na temat ich priorytetów, czyli kolejności wykonywania. Wiadomo np., że mnożenie jest „silniejsze” od dodawania, zatem najpierw mnożymy, potem dodajemy (kolejność tę można zmienić, stosując nawiasy okrągłe, dokładnie w taki sam sposób, w jaki zmienia się kolejność działań w matematyce). W JavaScriptcie jest podobnie — siła każdego operatora kolejność wykonywania jest ściśle określona.

9.2.3 Ćwiczenia

Napisz skrypt obliczający wartość wprowadzonego wyrażenia arytmetycznego. Do obliczeń wykorzystaj funkcję eval()

9.2.4 Opis założonych osiągnięć ucznia

Uczestnik zna podstawowe typy danych i operatory języka JavaScript oraz potrafi wykorzystać je w praktyce

9.3 Lekcja 3 – Instrukcje warunkowe i pętle

9.3.1 Cel lekcji.

Celem lekcji jest wprowadzenie instrukcji pozwalających na sterowanie wykonywaniem programu. Poznamy różne warianty instrukcji warunkowych oraz pętli oraz zobaczymy praktyczne przykłady ich wykorzystania. Instrukcje wprowadzone w tej lekcji są podstawą każdego języka programowania, bez ich znajomości nie można tworzyć żadnych złożonych programów.

9.3.2 Treść – slajdy z opisem

Slajd 1



W poprzedniej lekcji wprowadziliśmy pojęcia zmiennej oraz instrukcji. Dla przypomnienia – zmienna to symboliczna nazwa dla wartości przechowywanych w trakcie działania programu, a instrukcja to polecenie sprawdzające lub ustawiające jej stan. W tej lekcji pokażemy podstawowe instrukcje warunkowe w języku JavaScript



Slajd 2



Instrukcje warunkowe pozwalają na sterowanie przepływem programu wg. zdefiniowanych warunków. Możemy podzielić wykonywanie programu na różne ścieżki i w zależności od wartości zmiennych wybierać te, które powinny zostać wykonane.




Slajd 3



Instrukcje warunkowe

Wyróżniamy następujące rodzaje instrukcji:

- if
- if ... else
- if else if ...
- switch





Wyróżniamy następujące rodzaje instrukcji:

if
if ... else
if else if ...
switch


Pozwalają one na wykonywanie instrukcji w zależności od tego czy określony warunek został spełniony.

Slajd 4



Instrukcja if



```
if (warunek) {  
    instrukcje  
}  
lub jeśli jest tylko jedna instrukcja do wykonania  
if (warunek)  
    instrukcja;
```



wrażenia w nawiasach klamrowych zostaną wykonane tylko w przypadku, gdy warunek zostanie spełniony. Warunek musi być typu Boolean, można go zbudować za pomocą operatorów poznanych we wcześniejszej lekcji.

Nawiasy klamrowe wydzielają blok instrukcji wykonywanych po spełnieniu warunku. Po zamykającym nawiasie nie trzeba wstawiać średnika kończącego polecenie JavaScript. W przypadku gdy w wyniku spełnienia warunku należy wykonać tylko 1 instrukcję nawiasy można pominąć, ale wymagany wtedy jest ; na końcu linii.


Slajd 5



Instrukcja if

Przykład



- liczba = 21;
- if (liczba % 2 != 0) {
- document.writeln("Liczba nieparzysta");
- }



Na slajdzie widzimy przykład instrukcji warunkowej if. Warunkiem, który sprawdzamy jest fakt czy wartość zmiennej liczba jest podzielna przez 2. Jeśli nie to mamy do czynienia z liczbą nieparzystą i taki komunikat wpisujemy na stronie




Slajd 6





Instrukcja else

```
if (warunek) {  
    //instrukcje wykonywane, gdy warunek jest spełniony  
}  
else {  
    // instrukcje wykonywane gdy warunek nie jest  
    spełniony  
}
```



Instrukcja else rozszerza instrukcję if. Działa bardzo podobnie do if, ale zawarte w niej instrukcje są wykonywane tylko w przypadku gdy warunek nie jest spełniony.


Slajd 7



Instrukcja else



Przykład

- liczba = 21;
- if (liczba % 2 != 0) {
- document.writeln("Liczba nieparzysta");
- }
- else {
- document.writeln("Liczba parzysta");
- }




Na slajdzie jest rozszerzony przykład poprzedniej instrukcji. Poprzednim razem, jeśli liczba była podzielna przez 2 nic nie robiliśmy, program przechodził do wykonywania kolejnych instrukcji. Tym razem komunikat na stronie pojawi się w obu przypadkach. Będzie to albo napis "Liczba parzysta" lub "Liczba nieparzysta"

Slajd 8



Instrukcja else if



```
if (warunek1) {  
    instrukcje1;  
}  
else if (warunek2) {  
    instrukcje2;  
}  
.....  
else if (warunekN) {  
    instrukcjeN;  
}  
else {  
    instrukcjeM;  
}
```



Kolejna wersja instrukcji if pozwala na badanie wielu warunków. Po if może wystąpić wiele dodatkowych bloków else if.

Taka konstrukcja oznacza, że jeżeli warunek1 jest spełniony to wykonane zostaną instrukcje1, w przeciwnym wypadku gdy spełniony jest warunek2 to zostaną wykonane instrukcje2, itd. W przypadku, gdy wszystkie warunki są fałszywe, wykonywane są instrukcjeM z bloku else.


Slajd 9



Instrukcja else if



Przykład

- liczba = 21;
- if (liczba < 10) {
- document.writeln("Liczba mniejsza od 10");
- }
- else if (liczba > 20) {
- document.writeln("Liczba większa od 20");
- }
- else {
- document.writeln("Liczba z zakresu <10,20>");
- }




W tym przykładzie sprawdzamy w jakim przedziale znajduje się dana liczba. Na początku sprawdzamy czy liczba jest mniejsza niż 10 i jeśli tak to wyświetlamy stosowny komunikat. Jeśli liczba jest większa bądź równa 10 to sprawdzamy czy jest większa od 20. Jeśli tak to wypisujemy komunikat, że liczba jest większa od 20. Jeśli oba warunki nie zostały spełnione program przechodzi do wykonania instrukcji else – wypisujemy komunikat, że liczba jest z zakresu <10,20>

Slajd 10



Instrukcja switch



```
switch (wyrażenie) {  
  case wartość1:  
    instrukcja1;  
    break;  
  case wartość2:  
    instrukcja2;  
    break;  
  ...  
  default : instrukcja;  
}
```



Instrukcja wyboru switch (nazywana również instrukcją switch...case) pozwala w wygodny sposób sprawdzić ciąg warunków i wykonać różne instrukcje, w zależności od wyników porównywania.

Najpierw program szuka wartość odpowiedniego wyrażenia i wykonuje załączoną instrukcję. Jeśli nie uda się znaleźć odpowiedniej wartości program szuka domyślnej instrukcji i ją wykonuje. Jeśli wartość została znaleziona to program kontynuuje wykonywanie instrukcji aż do końca instrukcji **switch** lub do napotkania instrukcji **break**.


Slajd 11



Instrukcja switch

Przykład

- kolor = „Biały”
- switch (kolor) {
- case: „Zielony”: document.writeln(„Wybrano zielony kolor”);
- break;
- case: „Niebieski”: document.writeln(„Wybrano kolor niebieski”);
- break;
- case: „Biały”: document.writeln(„Wybrano kolor biały”);
- break;
- default: document.writeln(„Nieznany kolor”);
- }



W trakcie działania sprawdzane jest czy etykieta "Biały" znajduje się na zdefiniowanej liście etykiet instrukcji. Jeśli tak to wykonywanie programu przechodzi do tego miejsca. Jeśli pominiemy instrukcję break to oprócz napisu Wybrano kolor biały zobaczymy jeszcze napis Nieznany kolor.



Slajd 12

Slide 12 content: B2E logo, Szczeciński Park Naukowo-Technologiczny logo, and a blue vertical bar on the left. The main text is 'Ćwiczenie' followed by a bullet point: 'Napisz skrypt wyświetlający rozwiązania równania kwadratowego $Ax^2 + Bx + C$.'

Celem ćwiczenia jest praktyczne zapoznanie się z instrukcjami warunkowymi. Do tego celu wykorzystamy algorytm obliczania równania kwadratowego $Ax^2 + Bx + C$. Równanie ma rozwiązanie, jeśli parametr Δ (delta) równy $B^2 - 4 \cdot A \cdot C$ jest większy od 0 lub mu równy. Jeśli Δ równa się 0, mamy jedno rozwiązanie równe $-B/(2 \cdot A)$, jeśli Δ jest większa od 0, mamy dwa rozwiązania: $x_1 = (-B + \sqrt{\Delta})/(2 \cdot A)$ i $x_2 = (-B - \sqrt{\Delta})/(2 \cdot A)$.

Slajd 13

Slide 13 content: B2E logo, Szczeciński Park Naukowo-Technologiczny logo, and a blue vertical bar on the left. The main text is 'JavaScript' followed by 'Pętle'.

Instrukcje warunkowe nie są jedynymi sposobami na kontrolę przepływu działania programu. Jeśli konieczne jest powtórzenie dowolnego fragmentu kodu to wykorzystać możemy pętle.

Slajd 14



Slide 14 content: B2E logo, Szczeciński Park Naukowo-Technologiczny logo, and a blue vertical bar on the left. The main text is 'Rodzaje pętli w JavaScript' followed by a paragraph: 'Pętle są instrukcjami pozwalającymi na wielokrotne wykonywanie tego samego kodu. JavaScript obsługuje następujące rodzaje pętli:' followed by a list: 'for', 'while', 'do while'.

Pętle są instrukcjami pozwalającymi na wielokrotne wykonywanie tego samego kodu. JavaScript obsługuje następujące rodzaje pętli:

- for
- while
- do while




Slajd 15



Pętla for

Pętla for służy do wykonania instrukcji zawartych w nawiasach określonej liczbie razy.



```
for (wyrażenie_początkowe; warunek; wyrażenie_modyfikujące) {  
    instrukcje;  
}
```



Wyrażenie początkowe ($i = 0$) inicjuje zmienną używaną jako licznik pętli, warunek ($i < 5$) określa warunek którego spełnienie zakończy wykonywanie pętli, wyrażenie modyfikujące ($i++$) zwiększa lub zmniejsza licznik liczby wykonania. Powyższy przykład oznacza, dla zmiennej i równej początkowo 0 sprawdź, czy jest mniejsza od 5 i jeśli tak to wypisz jej wartość na wyjście a następnie podnieś wartość zmiennej i o 1 i ponownie sprawdź warunek. Kontynuuj aż warunek przestanie być prawdziwy.

Każde wykonanie pętli nazywamy iteracją lub przebiegiem, a zmienną zawierającą licznik pętli - zmienną iteracyjną.


Slajd 16



Pętla for

Przykład



```
for (i = 0; i < 5; i++) {  
    document.write(i + "<br />");  
}
```



Dla zmiennej iteracyjnej i równej początkowo 0 sprawdź warunek – czy liczba jest mniejsza od 5. jeśli tak to wypisz na ekranie wartość tej liczby a następnie zwiększ ją o 1.




Slajd 17



Pętla while



Pętla while służy do wielokrotnego wykonywania powtarzających się fragmentów kodu (na etapie pisania funkcji nie wiemy ile razy kod zostanie wykonany).

```
while (warunek) {  
    instrukcje;  
}
```



W przeciwieństwie do pętli for nie znamy z góry liczby iteracji w tej pętli. Pętla może wykonać dowolną ilość razy (dopóki warunek jest spełniony), instrukcje zawarte w pętli mogą również nie zostać wykonane, jeśli warunek nie został spełniony przed pierwszą iteracją. Musimy też sami zadbać o zainicjalizowanie zmiennej iterującej i jej zwiększanie bądź zmniejszanie w każdej iteracji pętli.


Slajd 18



Pętla while



Przykład

```
i = 0;  
while (i < 5) {  
    document.writeln(i + "<br/>");  
    i++;  
}
```



zmiennej i przypisz wartość 0. Dopóki i jest mniejsze od 5 wypisuj wartość i na ekranie a następnie zwiększ i o 1


Slajd 19



Pętla do while

W pętli do ... while warunek sprawdzany jest po wykonaniu zestawu instrukcji.



```
do {  
    instrukcje;  
}  
while (warunek);
```



Specjalnym rodzajem pętli while jest pętla do while. Różnicą w porównaniu do pętli while jest fakt, że instrukcje zawsze zostaną wykonane przynajmniej raz - nawet jeśli warunek początkowy nie jest spełniony.




Slajd 20





Pętla do while

Przykład:

```
i = 0
do {
  document.writeln(i);
  i++;
} while (i<5)
```




Slajd 21





Zagnieżdżanie pętli

- W języku JavaScript definicje pętli możemy umieszczać wewnątrz innych pętli. np.
- ```
for (i = 0; i < 5; i++) {
```
- ```
  document.writeln("Jestem w pętli zewnętrznej" + i);
```
- ```
 for (j = 0; j < 5; j++) {
```
- ```
    document.writeln("Jestem w pętli wewnętrznej „+j”);
```
- ```
 }
```
- ```
}
```




Slajd 22



Zagnieżdżanie pętli

wynik:

```
Jestem w pętli zewnętrznej 1
Jestem w pętli wewnętrznej 1
Jestem w pętli wewnętrznej 2
Jestem w pętli zewnętrznej 2
Jestem w pętli wewnętrznej 1
Jestem w pętli wewnętrznej 2
Jestem w pętli zewnętrznej 3
Jestem w pętli wewnętrznej 1
Jestem w pętli wewnętrznej 2
```



Jak widzimy zagnieżdżanie pętli działa w następujący sposób:
dla każdej iteracji pętli zewnętrznej wykonywane są wszystkie iteracje pętli wewnętrznej



Slajd 23

Slide 23 content: A slide titled 'Ćwiczenie' (Exercise) with a bullet point: 'Wykorzystując przykład podany na poprzednim slajdzie napisz funkcję wyświetlającą na ekranie tabliczkę mnożenia' (Using the example given on the previous slide, write a function displaying a multiplication table on the screen). The slide includes logos for B2E, Szczeciński Park Naukowo-Technologiczny, and DAILY GROUP.

Slajd 24

Slide 24 content: A slide titled 'Ćwiczenie' (Exercise) showing a JavaScript code snippet for a multiplication table. The code is:

```
for (i = 1; i <= 10; i++) {  
  for (j = 1; j <= 10; j++) {  
    document.write(i + " * " + j + " = " + i * j + "<br />");  
  }  
  document.write("<br />");  
}
```

 The slide includes logos for B2E, Szczeciński Park Naukowo-Technologiczny, and DAILY GROUP.

9.3.3 Ćwiczenia

Ćwiczenia zostały przedstawione na slajdach 12 oraz 23-24.

W pierwszym ćwiczeniu zwracana jest uwaga na wykorzystanie instrukcji warunkowej if w celu napisania programu obliczającego rozwiązanie równania kwadratowego.

Drugie ćwiczenie umożliwia praktyczne rozpoznanie sposobu działania pętli – wypisujemy w nim na ekran wartości z tabliczki mnożenia.

9.3.4 Opis założonych osiągnięć ucznia

Uczestnik po zakończeniu tej lekcji zna instrukcje warunkowe i pętle i potrafi wykorzystać je w praktyce.

9.4 Lekcja 4 – Tablice

9.4.1 Cel lekcji

Celem lekcji pokazanie, czym są i do czego można wykorzystywać zmienne tablicowe w języku JavaScript.

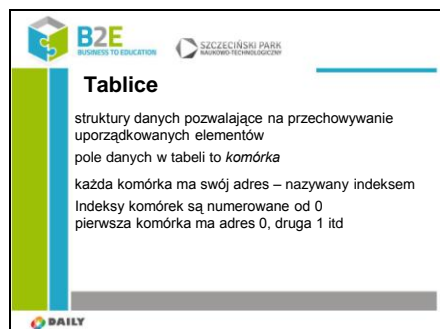
9.4.2 Treść – slajdy z opisem

Slajd 1



W poprzednich lekcjach korzystaliśmy tylko z pojedynczych zmiennych. Często jednak potrzebujemy mechanizmu, który pozwoli nam na pogrupowanie zmiennych, albo nawet na zmienną do której możemy wpisać więcej niż jedną wartość. Do tego celu służą tablice.

Slajd 2





W JavaScriptcie możemy zarządzać danymi w sposób bardziej zaawansowany niż za pomocą zmiennych. Jednym z przykładów są tablice – można powiedzieć, że są to kolekcje zmiennych uporządkowane i pogrupowane razem.

Poszczególne elementy w tabeli nazywamy komórkami, adres każdej komórki indeksem.

Prostą tablicę można sobie wyobrazić jako szafkę, w której każda szuflada jest komórką a numer tej szuflady indeksem komórki. Indeksy w tabelach JavaScript numerowane są od 0 – pierwsza komórka w tabeli ma indeks 0, druga 1 itd..


Slajd 3



Tablice


Tworzenie nazw tablic

- rozpoczyna się od małej lub wielkiej litery, znaku_ lub znaku \$
- nie może zawierać operatorów matematycznych (+ -), znaków punktacyjnych (takich jak !, &) albo spacji
- na drugim i dalszych miejscach może zawierać cyfry
- słowa kluczowe języka nie mogą być użyte jako nazwa



Tworzenie nazw tablic rozpoczyna się od małej lub wielkiej litery, znaku_ lub znaku \$ nie może zawierać operatorów matematycznych (+ -), znaków punktacyjnych (takich jak !, &) albo spacji na drugim i dalszych miejscach może zawierać cyfry słowa kluczowe języka nie mogą być użyte jako nazwa


Slajd 4



Tworzenie tablic



Dwa sposoby

- przez deklarację zmiennej w nawiasach kwadratowych
`var tablica = ["Janek", "Franek", "Bronek"];`
- przez wywołanie konstruktora tablicy
`var tablica = new Array("Janek", "Franek", "Bronek");`




Tablicę możemy utworzyć na dwa sposoby – za pomocą nawiasów kwadratowych podając wewnątrz nawiasów komórki tablicy, lub wywołując konstruktor obiektu Array, który jest reprezentacją tablicy w JavaScript (o obiektach i konstruktorach powiemy sobie później). Efektem działania obu instrukcji będzie zmienna tablica zawierająca 3 imiona – Janek, Franek i Bronek.

Slajd 5





Przykłady tworzenia tablic

```
var tablica = [1,2,3,4,5,6,7,8,9,10];  
var tablica = [1, "Janek", 2, "Franek"];  
var tablica = ["Janek", "Franek", "Bronek"];  
var tablica = [];
```



Wartość każdej komórki może być dowolnego typu. Przykładowo możemy utworzyć tablicę zawierającą liczby całkowite od 1 do 10, wcześniej widzieliśmy przykład deklaracji tablicy zawierającej ciągi znaków. W tablicy możemy też umieścić elementy różnego typu – np. liczby i ciągi znaków. Notacja z wykorzystaniem nawiasów kwadratowych umożliwia pomijanie elementów w tablicy – komórki pominięte będą miały wartość undefined. Możemy także utworzyć pustą tablicę i dodawać do niej elementy w trakcie działania programu.

Slajd 6



 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY**Przykłady tworzenia tablic**

```
var tablica = new Array();  
var tablica = new Array(1,2,3,4,5,6,7,8,9,10);  
var tablica = new Array(5);
```

w konstruktorze tablicy możemy nie podawać żadnych argumentów – zostanie wtedy utworzona pusta tablica, możemy podać argumenty oddzielone przecinkami – zostaną one wtedy wpisane jako wartości komórek tablicy.

Trzecia przykładowa konstrukcja ma trochę inne działanie. Zamiast spodziewanego utworzenia tablicy z jedną komórką o wartości 5, zostanie utworzona tablica pięcioelementowa w której komórki będą miały wartość undefined

Slajd 7

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY**Odczyt i zapis danych**

```
var zmienna = tablica[5];  
document.write(tablica[5]);  
if (tablica[5] == wartość) {  
    instrukcje;  
}  
tablica[5] = wartość;  
var indeks = 6;  
tablica[indeks] = nowa_wartość;
```

Odczyt danych z tablicy jest prosty. Aby przypisać wartość z danej komórki tablicy wystarczy podać jej indeks.

Nie musimy oczywiście korzystać z dodatkowych zmiennych aby wyświetlić lub wykorzystać wartość komórki. Możliwe jest bezpośrednie operowanie na komórkach tabeli.

Aby zapisać wartość do komórki tablicy wykonujemy operację przypisania na tablicy z wykorzystaniem indeksu komórki, którą chcemy zmodyfikować.

Wartości indeksu nie musimy podawać w trakcie pisania kodu – możemy wykorzystać do tego zmienną i ustawić wartość w trakcie działania

programu.

Możemy jako indeks tablicy
wykorzystać komórkę innej tablicy

Slajd 8

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY

Tablice asocjacyjne

Rozważmy taką tablicę:

```
var wypłaty = new Array(„Janek”, 2000, „Franek”, 3000,  
„Bronek”, 2500);  
var wypłaty = new Array();  
wypłaty[„Janek”] = 2000;  
wypłaty[„Franek”] = 3000;  
wypłaty[„Bronek”] = 2500;
```



Używanie indeksów numerycznych jest dobre, jeśli operujemy na niewielkiej liczbie danych.

Jeśli danych jest więcej lub jeśli dane mają dodatkowe znaczenie, użycie numerów aby pobrać wartość z tabeli zaczyna robić się kłopotliwe.



Rozważmy tablicę z wypłatami w następującej postaci: w pierwszej komórce jest identyfikator osoby (imię) a w kolejnej jego zarobki. Możliwe jest wyszukanie zarobków Franka w następujący sposób – sprawdź, czy wartość pierwszej komórki to „Franek” – jeśli tak to pobierz i wyświetl wartość kolejnej komórki. W przeciwnym wypadku przeskocz o 2 komórki i wykonaj sprawdzenie ponownie.

Na szczęście javascript wspiera przypisywanie nazw elementom komórek. Możemy zatem utworzyć następującą tablicę, gdzie jako indeksy pól podamy identyfikatory osób a jako wartość ich wypłatę. Wyświetlenie dochodów Franka to w tym przypadku proste zapytanie `wypłaty[„Franek”]`.

JavaScript nie tworzy w tym przypadku nowych elementów tablicy tylko dodaje nowe właściwości (pary typu klucz – wartość) do obiektu przechowującego tablicę. Zapytanie o długość tablicy zwróci w tym przypadku 0 – bo utworzyliśmy pustą tablicę i dodaliśmy do niej 3

właściwości

Slajd 9

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY

Tablice asocjacyjne

Tworzenie tablicy asocjacyjnej z wypełnieniem danymi

```
var wypłaty = new Array(
```

```
{
```

```
  „Janek” : 2000,
```

```
  „Franek” : 3000,
```

```
  „Broniek” : 2500
```



```
});
```

 **DAILY**
GROUP

Tworząc tabelę asocjacyjną zainicjowaną początkowym zestawem danych do konstruktora obiektu przekazujemy definicję wartości, które chcemy utworzyć.

Definicja zawiera pary klucz wartość (oddzielone dwukropkiem, a nie znakiem równości) umieszczone w nawiasach klamrowych

Slajd 10

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY

Ćwiczenie 1

Utwórz tablicę przechowującą liczby od 1 – 100 w taki sposób, że w komórkach od 0-49 będą wartości 1-50 a w komórkach od 50-99 wartości 100 – 51.

Wyświetl zawartość tabeli na ekranie.

 **DAILY**
GROUP



Ćwiczenie:

Utwórz tablicę przechowującą liczby od 1 – 100 w taki sposób, że w komórkach od 0-49 będą wartości 1-50 a w komórkach od 50-99 wartości 100 – 51.

Wyświetl zawartość tabeli na ekranie.




Slajd 11



Ćwiczenie 1

```
for (i = 0; i < 100; i++) {  
  if (i < 50) {  
    tablica[i] = i + 1;  
  }  
  else {  
    tablica[i] = 100 - (i - 50);  
  }  
}  
for (i = 0; i < 100; i++) {  
  document.write(tablica[i] + "<br />");  
}
```



Slajd 12





JavaScript

Tablice wielowymiarowe



Slajd 13




Tablice wielowymiarowe

```
var tablica = [1,2,3];  
tablica[x]  
  
var tablica = [  
  ["A1", "B1", "C1"],  
  ["A2", "B2", "C2"]  
];  
tablica[x][y]
```

1	2	3
---	---	---

A1	B1	C1
A2	B2	C2



Do tej pory wszystkie pokazane tablice miały tylko 1 wymiar. Można je było przedstawić za pomocą wektora z wartościami komórek tabeli.

Do pojedynczej wartości można było dostać się podając jej indeks.

Czy możliwe jest zatem podanie więcej niż jednego indeksu, aby wskazać element tablicy?



JavaScript nie wspiera niestety wielowymiarowych tablic, ale jest sposób na zasymulowanie takiej funkcjonalności.

W języku JavaScript wartością komórek tablicy mogą być inne tablice. Możemy utworzyć tabele zawierające w komórkach inne tabele, które mogą zawierać kolejne tabele itd.


Dostęp do elementów w takich

tabelach jest następujący: `tablica[x][y]`
– Z tablicy pobierz wartość komórki oznaczonej indeksem x a z niej wartość komórki oznaczonej Y

Slajd 14

 
Użycie pętli for

```
var tablica = [];  
for (var i = 0; i < 100; i++) {  
  tablica[i] = „wartość „+i;  
}  
for (var i = 0; i < tablica.length; i++) {  
  document.write(tablica[i] + „<br />”);  
}
```



Bezpośrednie odwołania do elementów tablicy za pomocą indeksu są wystarczające tylko w przypadku, gdy tablica zawiera niewiele elementów. Jeśli danych jest dużo, lub z góry nie wiemy ile (częsta sytuacja, gdy nasz program otrzyma dane z zewnątrz) wymagane będzie wprowadzenie automatycznego przetwarzania tablicy. W takim celu możemy wykorzystać pętle.

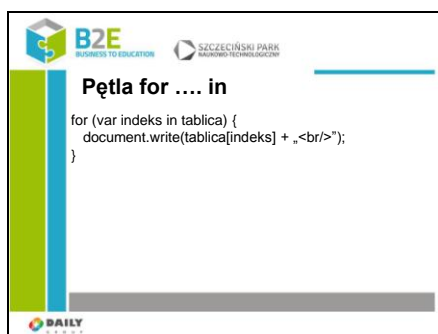
Pierwsza przedstawiona pętla dodaje 100 elementów do nowo utworzonej tablicy.

Odczyt danych z tablicy możliwy jest w następujący sposób.

Pętla for może zostać wykorzystana w przypadku gdy znamy długość tablicy. W przypadku tablic z indeksem numerycznym długość tablicy możemy pobrać za pomocą właściwości `length`. Właściwość ta przyjmuje wartość najwyższego indeksu w tablicy + 1.



Slajd 15



Slide 15 content: A presentation slide titled "Pętla for in" (for loop). It contains a code snippet:

```
for (var indeks in tablica) {  
    document.write(tablica[indeks] + "<br/>");  
}
```

 The slide has a blue header with logos for B2E, Szczeciński Park Naukowo-Technologiczny, and DAILY GROUP.

Jeśli nie znamy długości tablicy, lub jeśli wiemy że zawiera ona komórki niezdefiniowane możemy wykorzystać pętlę specjalną for in. Definicja pętli jest następująca:

```
for (var indeks in tablica) {  
    instrukcje  
}
```

Wynikiem działania funkcji jest przejście po wszystkich zdefiniowanych indeksach (funkcja wyświetli tylko komórki mające wartość) oraz dodanych do obiektu właściwościach.

Slajd 16



Slide 16 content: A presentation slide titled "Operacje na tablicach" (Operations on arrays). It lists several methods:

- Łączenie tablic (concat)
- Odwroćenie kolejności elementów (reverse)
- Dodawanie i usuwanie elementów (pop, push, shift, unshift)
- sortowanie (sort)
- sklejanie (join)


 The slide has a blue header with logos for B2E, Szczeciński Park Naukowo-Technologiczny, and DAILY GROUP.

Tablice są obiektami w JavaScript, zawierającymi kilka predefiniowanych metod umożliwiających m.in:

- concat. Tablice zostaną dodane do siebie a powstały wynik zwrócony w wyniku działania funkcji
- reverse – odwraca kolejność elementów w tablicy
- pop – pobiera i usuwa z tablicy ostatni element
- push - dodaje na końcu tablicy nowy element
- shift – pobiera i usuwa pierwszy element z tablicy
- unshift – wstawia element na początku tablicy (odwrotność shift)
- sort – sortuje tablicę. Jako element należy podać nazwę funkcji porównującej 2 elementy
- join – zwraca łańcuch znaków z zawartością wszystkich komórek oddzielonych znakiem separatora




Slajd 17



Łączenie tablic

```
tablica2 = tablica.concat( lista elementów)

var dni_tygodnia = ["Poniedziałek", "Wtorek", "Środa",
"Czwartek", "Piątek", "Sobota", "Niedziela"];
var miesiace = ["Styczeń", "Luty", "Marzec", "Kwiecień"];
var wynik = dni_tygodnia.concat(miesiace);
```



Wynikiem połączenia tablic jest nowa tablica składająca się z wszystkich elementów tablicy dni_tygodnia oraz wszystkich elementów tablicy miesiace

Slajd 18



Odwracanie tablic

```
var tablica = [1,2,3,4,5];
tablica.reverse();
wynik: [5,4,3,2,1]
```



Do odwracania tablic służy funkcja reverse. Nie przyjmuje ona żadnych argumentów i w działa na tablicy, na rzecz której została wywołana (nie jest tworzona tablica wynikowa)

Slajd 19



Dodawanie i usuwanie elementów

- pop – tablica.pop()
- push – tablica.push(lista elementów)
- shift – tablica.shift()
- unshift – tablica.unshift(lista elementów)



Metoda pop pobiera ostatni element z tablicy i zwraca jego wartość. Tablica zostaje zmniejszona o 1.


Metoda push dodaje elementy przekazane jako parametr na końcu tablicy

Metoda shift działa tak samo jak pop – tylko że pobiera pierwszy element z tablicy.

Metoda unshift dodaje listę elementów na początku tablicy. Tablica zostanie wtedy od nowa przenumерована



Slajd 20



Sortowanie tablicy

Aby posortować tablicę należy wywołać na niej funkcję `sort()`

```
var liczby = [5,7,3,5,2,7,5];  
liczby.sort();  
wynik: [2,3,5,5,7,7]
```



Sortowanie tablicy to uporządkowanie jej elementów. Funkcja zmienia porządek elementów w tablicy nadpisując oryginalną wersję.

Tablice zawierające liczby zostaną uporządkowane rosnąco, natomiast tablice zawierające ciągi znaków – alfabetycznie.

Slajd 21



Sortowanie cd.

`tablica.sort(nazwa_funkcji);`
Funkcja musi zwracać:

- wartość mniejszą od 0, jeśli pierwszy argument jest mniejszy od drugiego
- wartość 0, jeśli argumenty są równe,
- wartość większą od 0, jeśli pierwszy argument jest większy od drugiego



Jeśli będziemy chcieli posortować tablice w sposób odmienny od domyślnego do wywołania metody sort należy przekazać nazwę funkcji porównującej 2 elementy.

Taka funkcja będzie otrzymywać 2 elementy tablicy, jako wynik musi zwracać:

- ☐ wartość¹ mniejszą od 0, jeżeli pierwszy argument jest mniejszy od drugiego
- ☐ wartość¹ równą 0, jeżeli argumenty są równe
- ☐ wartość¹ większą od 0, jeżeli pierwszy argument jest większy od drugiego

Aby odwrócić kolejność sortowania możemy albo zmodyfikować funkcję porównującą, albo na wyniku sortowania zastosować metodę `reverse()`



Slajd 22

Sklejanie wartości (join)

```
var tablica = ["Janek", "Franek", "Bronek"];  
document.write(tablica.join(" i "));
```

Janek i Franek i Bronek
Janek,Franek,Bronek

Czasem przydatna jest możliwość zmiany wszystkich elementów tablicy w jeden łańcuch znaków. Można do tego wykorzystać metodę `join()` jako argument podając separator rozdzielający poszczególne elementy tablicy. W podanym przykładzie jako separator wykorzystaliśmy łańcuch składający się ze znaku spacji na początku i końcu oraz spójnika `i`. Wynikiem działania skryptu będzie tekst: Janek i Franek i Bronek. Jeśli nie podamy żadnego separatora domyślnie zostanie przyjęty znak `,`

9.4.3 Ćwiczenia

Ćwiczenie zostało zaprezentowane na slajdzie 10. Utwórz tablicę przechowującą liczby od 1 – 100 w taki sposób, że w komórkach od 0-49 będą wartości 1-50 a w komórkach od 50-99 wartości 100 – 51. Ćwiczenie ma na celu pokazanie sposobu indeksowania wartości w tablicy.

9.4.4 Opis założonych osiągnięć ucznia

Po ukończeniu lekcji uczestnik będzie znał sposoby tworzenia tablic w języku JavaScript, a także podstawowe metody służące do pracy z tablicami

9.5 Lekcja 5 – Obiekty i funkcje

9.5.1 Cel lekcji

Język JavaScript jest językiem obiektowym – w tej lekcji poznamy definicję obiektu oraz funkcji.

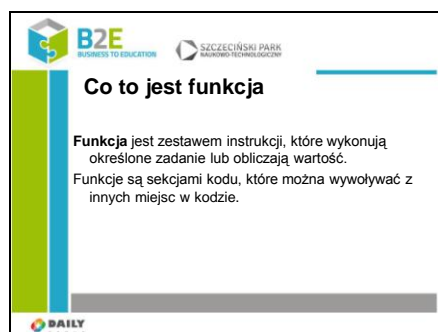
9.5.2 Treść – slajdy z opisem

Slajd 1



We wcześniejszych lekcjach wprowadziliśmy pojęcia zmiennej, pokazaliśmy istniejące typy danych w języku JavaScript i omówiliśmy tablice jako specjalny rodzaj zmiennych. JavaScript pozwala nam na tworzenie własnych typów danych zwanych obiektami i zbiorów instrukcji, które mogą wykonywać jakąś część programu zwanych funkcjami. W tej lekcji przybliżymy te pojęcia.

Slajd 2

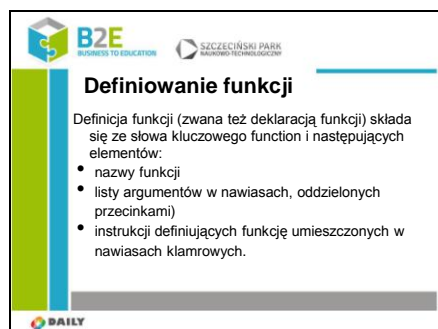


Funkcja to jeden z podstawowych bloków w JavaScript. Funkcja jest zestawem instrukcji, które wykonują określone zadanie lub obliczają wartość.

Funkcja wykonuje zdefiniowane w ciele funkcji instrukcje a następnie zwraca wynik do miejsca, z którego została wywołana.

Aby wykorzystywać funkcje trzeba ją wcześniej zdefiniować.

Slajd 3



Definicja funkcji (zwana też deklaracją funkcji) składa się ze słowa kluczowego function i następujących elementów:

- nazwy funkcji
- listy argumentów w nawiasach, oddzielonych przecinkami)
- instrukcji definiujących funkcję umieszczonych w nawiasach klamrowych.

Nazwa może się składać z liter, cyfr oraz znaków podkreślenia i dolara, nie może jednak zaczynać się od cyfry. We wszystkich współczesnych implementacjach JavaScriptu może też zawierać znaki spoza alfabetu łacińskiego (np. polskie znaki

Moduł szkoleniowy JavaScript str. 47

diakrytyczne), choć z reguły się ich nie stosuje.

Funkcja może, ale nie musi zawierać słowa kluczowego `return` (zwracającego wartość funkcji). Gdy w definicji funkcji brakuje instrukcji `return` wartością zwracaną przez funkcję jest "undefined".

Slajd 4

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY

Definiowanie funkcji

```
function nazwa(argument1, argument2, ... , argumentN)
{
    //instrukcje
}

function kwadrat(liczba) {
    return liczba * liczba;
}
```

 **DAILY**
GROUP

Funkcja posiada 1 argument nazwany `liczba` i zwraca za pomocą instrukcji `return` wartość argumentu pomnożonego przez samego siebie. Typy podstawowe (np. `number`) są przekazywane do funkcji za pomocą wartości. Oznacza to, że jeśli funkcja zmodyfikuje wartość parametru to zmiana ta nie będzie widoczna poza funkcją, przekazanie do funkcji typu złożonego (np. tablicy, lub zdefiniowanego przez użytkownika obiektu) spowoduje przekazanie tylko referencji do obiektu. Wszystkie modyfikacje wykonane przez funkcje będą widoczne także poza nią.

Slajd 5

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY

Argumenty funkcji

- brak z góry zdefiniowanej liczby
- służą do przekazania wartości do funkcji w celu dowolnego przetwarzania i wykonywania na nich instrukcji



 **DAILY**
GROUP

Argumenty funkcji

- brak z góry zdefiniowanej liczby
- służą do przekazania wartości do funkcji w celu dowolnego przetwarzania i wykonywania na nich instrukcji




Slajd 6



Zwracanie wartości przez funkcję



- słowo kluczowe **return**

```
function nazwa (argumenty)
{
    //instrukcje
    return wartość;
}
```



Jeśli w ciele funkcji wystąpi instrukcja **return** to przetwarzanie funkcji zostaje przerwane i do miejsca wywołania zwracana jest wartość występująca po słowie **return**

Slajd 7




Zasięg (widoczność) zmiennych

Zasięg zmiennej to miejsce w którym zmienna jest ważna i można się do niej bezpośrednio odwoływać.

Zmienne mogą być:

- globalne,
- lokalne



Kiedy zadeklarujemy zmienną poza ciałem funkcji jest ona nazywana zmienną globalną, ponieważ jest dostępna z poziomu każdego innego fragmentu kodu w bieżącym dokumencie,

Jeśli zadeklarujemy zmienną w ciele funkcji jest ona nazywana zmienną lokalną - widoczną tylko w obrębie tej funkcji (i ewentualnych podfunkcji w niej zdefiniowanych)

Aby zadeklarować zmienną lokalną należy poprzedzić ją słowem kluczowym **var**. Wartość takiej zmiennej zostanie zapomniana po opuszczeniu funkcji, a jeśli istniała wcześniej zmienna globalna o takiej samej nazwie, zachowa ona swoją wartość sprzed wywołania funkcji.

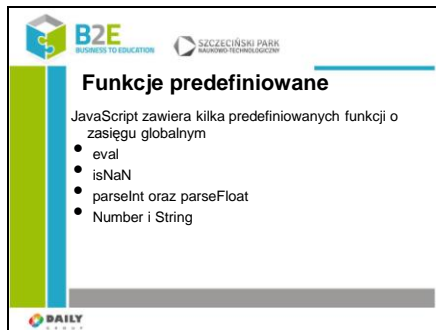
Zmienna zadeklarowana w funkcji bez wykorzystania słowa kluczowego **var** jest zmienną globalną.



Slajd 8



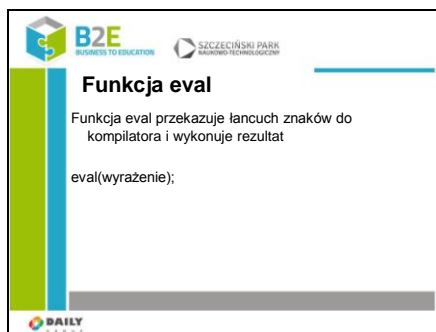
Slajd 9



JavaScript zawiera kilka predefiniowanych funkcji o zasięgu globalnym

- eval
- isNaN
- parseInt oraz parseFloat
- Number i String

Slajd 10



Funkcja eval przekazuje łańcuch znaków do kompilatora i wykonuje rezultat



`eval(wyrażenie);`

Nie zaleca się nadmiernie wykorzystywać tej funkcji, ponieważ utrudnia, a czasem nawet uniemożliwia sprawdzanie poprawności kodu z wykorzystaniem narzędzi typu Jslint. Dodatkowo funkcja eval zużywa czas na przekształcenie wyrażenia.





Slajd 11




Funkcja eval - przykłady

```
eval("x=10;y=20;document.write(x*y)");  
document.write("<br>" + eval("2+2"));  
document.write("<br>" + eval(x+17));
```



wynikiem będą:

200
4
27



```
eval("x=10;y=20;document.write(x*y)");  
;  
document.write("<br>" + eval("2+2"));  
document.write("<br>" + eval(x+17));  
wynikiem będą:  
200  
4  
27
```


Slajd 12



Funkcja isNaN



Sprawdza czy wartość jest liczbą.
isNaN(wartość);

Zwraca true jeśli przekazana wartość jest "NaN"



Funkcje parseFloat i parseInt zwracają NaN, gdy podana jako argument wartość nie jest liczbą. Wykorzystując funkcję sprawdzającą isNaN możemy w kodzie programu sprawdzić czy konwersja się udała i w zależności od wyniku odpowiednio zareagować

Slajd 13




Funkcja isNaN - przykłady

```
document.write(isNaN(5-2)+ "<br>");  
document.write(isNaN(0)+ "<br>");  
document.write(isNaN("Hello")+ "<br>");  
document.write(isNaN("2005/12/12")+ "<br>");
```

wynikiem będzie:

false
false
true
true





Slajd 14



Funkcje parseInt i parseFloat

Funkcje konwertujące łańcuch znaków na liczby całkowite i zmiennoprzecinkowe.



```
parseFloat(str);  
parseInt(str [, base]);
```



Funkcja `parseFloat` próbuje zwrócić liczbę zmiennoprzecinkową z podanego łańcucha znaków. Jeśli natrafi na znak inny niż `+` lub `-`, cyfrę(0-9) lub `.` zwraca wartość uzyskaną do danego momentu i ignoruje resztę ciągu. Jeśli pierwszy znak w łańcuchu nie może być skonwertowany do liczby funkcja zwraca `NaN`.


Funkcja `parseInt` konwertuje pierwszy argument (`str`) na liczbę całkowitą zgodną z wybraną bazą. Np. baza 10 zwróci liczbę jako dziesiętną 8 jako ósemkową, 16 - szesnastkową itd. Jeśli podczas przetwarzania funkcja trafi na znak nie znajdujący się w podanej bazie przerwie wykonywanie i zwróci wartość uzyskaną do tego momentu. Jeśli pierwszy znak w ciągu nie może być zmieniony na liczbę funkcja zwraca `NaN`.

Slajd 15





parseFloat - przykłady

```
document.write(parseFloat("10") + "<br>");  
document.write(parseFloat("10.33") + "<br>");  
document.write(parseFloat("34 45 66") + "<br>");  
document.write(parseFloat(" 60 ") + "<br>");  
document.write(parseFloat("40 lat") + "<br>");  
document.write(parseFloat("On ma 40 lat") + "<br>");
```






Slajd 16





parseFloat - przykłady cd.

wynikiem będą:

```
10  
10.33  
34  
60  
40 (!!!)  
NaN
```




Slajd 17



parseInt - przykłady

```
document.write(parseInt("10") + "<br>");  
document.write(parseInt("10.33") + "<br>");  
document.write(parseInt("34 45 66") + "<br>");  
document.write(parseInt("40 lat") + "<br>");  
document.write(parseInt("On ma 40 lat") + "<br>");  
document.write(parseInt("10", 10) + "<br>");  
document.write(parseInt("010") + "<br>");  
document.write(parseInt("10", 8) + "<br>");  
document.write(parseInt("0x10") + "<br>");  
document.write(parseInt("10", 16) + "<br>");
```





Tylko pierwsza liczba w łańcuchu jest zwracana,
spacje na początku i końcu wyrażenia są dopuszczalne

Starsze przeglądarki ustawiały domyślną bazę na ósemkową jeśli łańcuch będzie rozpoczynał się od 0.

W wersji ECMAScript 5 domyślną bazą są liczby dziesiętne


Slajd 18



parseInt - przykłady cd.



wyniki:

```
10  
10  
34  
40 (!!!)  
NaN  
10  
8 (!!!)  
8  
16  
16
```






Slajd 19



Funkcja Number i String

Funkcje pozwalające na konwersję obiektu do liczby lub łańcucha znaków

```
var obiekt;  
obiekt = Number(obiekt);  
obiekt = String(obiekt);
```



Funkcja Number wykorzystuje metodę valueOf() obiektu a funkcja String metodę toString().

Slajd 20





JavaScript

Obiekty




Slajd 21



Czym jest obiekt?

Obiekt:



- dowolny byt, który chcemy zapisać w pamięci komputera
- posiada właściwości, będące parami klucz i wartość
- może posiadać metody - funkcje operujące na obiekcie



Obiekty są kolekcjami właściwości, gdzie każda właściwość ma nazwę oraz wartość. Nazwa może być dowolnym łańcuchem, również pustym. Wartość właściwości może być dowolną, dozwoloną wartością JavaScriptu, z wyjątkiem undefined. Obiekty JavaScriptu są bezklasowe. Nie ma żadnych ograniczeń co do nazw nowych właściwości lub ich wartości. Obiekty są wygodne do przechowywania i organizowania danych. Obiekty mogą zawierać inne obiekty.




Slajd 22





Obiekty w JavaScript

- zmienne
- funkcje
- tablice
- obiekty zdefiniowane przez użytkownika



W języku JavaScript proste typy to liczby, łańcuchy, typy logiczne oraz typy specjalne (null i undefined). Wszystkie pozostałe elementy, wliczając funkcje, zmienne, tablice to obiekty


Slajd 23



Tworzenie prostych obiektów

- JSON - JavaScript Object Notation
- obiekt należy umieścić między nawiasami klamrowymi



```
{  
  definicja obiektu  
}
```



do tworzenia prostych obiektów można wykorzystać notację JSON (JavaScript Object Notation) - notacja obiektów JavaScript. Definicję obiektu należy umieścić w nawiasach klamrowych. Definicja składa się z właściwości i przypisanych im wartości.


Nazwa właściwości powinna być ciągiem znaków w cudzysłowie, natomiast wartość może być łańcuchem znaków, liczbą, obiektem, tablicą itd. Kolejne właściwości oddzielamy od siebie przecinkiem.

Slajd 24



JSON - przykład



```
{  
  "producent": "Fiat",  
  "model": "Punto",  
  "rocznik": 2010  
}
```



Powyższa definicja opisuje przykładowy obiekt samochód. Zawiera 3 właściwości: producent, model i rocznik.




Slajd 25



Obiekty JSON w skryptach

```
var obiekt = {  
  //definicja  
}  
np.  
var samochod = {  
  "producent": "Fiat",  
  "model": "Punto"  
}
```



Obiekt JSON można przypisać do zmiennej albo wykorzystać funkcję eval

Slajd 26




Obiekty JSON w skryptach

```
var zmienna = nazwaObiektu.nazwaWłaściwości  
np.  
  
var model_samochodu = samochod.model;
```



po utworzeniu obiektu możemy uzyskać dostęp do jego właściwości. W tym celu korzystamy z operatora . (znak kropki)

Slajd 27



Bezpośrednie przypisywanie wartości

Możemy przypisać wartości do obiektu już po jego utworzeniu, np.



```
samochod.kolor = "Biały";  
  
samochod["cena"] = 50000;
```



Jeśli przy operacji przypisania odwołamy się do nieistniejącej właściwości zostanie ona utworzona. Dostęp do właściwości obiektu możliwy jest za pomocą . lub nawiasu klamrowego. W nawiasie podajemy nazwę właściwości, której wartość chcemy zmodyfikować




Slajd 28



Odczyt i zapis za pomocą pętli



```
for ( var nazwa in nazwaObiektu ) {  
    instrukcje  
}
```



Specjalna wersja pętli for działająca na właściwościach obiektu. Za każdą iteracją pod zmienną nazwa zostanie podstawiona kolejna właściwość obiektu nazwaObiektu. Pętla będzie wykonywana aż odczytane zostaną wszystkie właściwości, lub zostanie przerwana za pomocą instrukcji return lub break;

Pętla pozwala w prosty sposób przejść po właściwościach obiektu bez konieczności posiadania informacji o ich nazwach i liczbie.


Slajd 29



Metody obiektów

Metoda to właściwość obiektu, której wartością jest funkcja

```
obiekt.nazwa_funkcji = function() {  
    instrukcje  
}
```



Metody wykonują zwykłe operacje na danych zawartych w danym obiekcie, więc muszą mieć dostęp do jego właściwości. Służy do tego słowo kluczowe this. Odróżnia się w ten sposób właściwość obiektu od innych zmiennych

Slajd 30



Metody cd.



```
var obiekt = {  
    //definicje właściwości  
    nazwa_funkcji : function () {  
        instrukcje  
    }  
}
```



Definicje metody można umieścić w notacji JSON, tak samo jak każdą inną właściwość obiektu. W ten sposób możemy utworzyć obiekt zawierający zarówno właściwości jak i funkcje. Metody obiektu mogą przyjmować argumenty, definiujemy je tak samo jak argumenty w funkcji./




Slajd 31



Metody - przykład

```
var samochod = {  
  "producent": "Fiat",  
  "model": "Punto"  
  "wyswietl": function() {  
    alert(this.producent + " " + this.model);  
  }  
}
```



Tworzymy obiekt samochód zawierający właściwości producent i model oraz metodę wyświetlającą komunikat z informacją o wartościach właściwości obiektu

Slajd 32




JavaScript

Konstruktory i prototypy




Slajd 33



Konstruktory


```
function Samochod(producent, model, rocznik) {  
  this.producent = producent;  
  this.model = model;  
  this.rocznik = rocznik;  
}  
var samochod = new Samochod("Fiat", "Punto", 2010);
```



Tworzenie obiektów za pomocą JSON nie jest jedyną dostępną możliwością. Możemy również utworzyć obiekt za pomocą operatora new, której jako parametr podamy specjalną funkcję zwaną konstruktorem.

Operator new tworzy nowy pusty obiekt a następnie przekazuje go funkcji będącej jego argumentem. Zadaniem tej funkcji jest zainicjalizowanie obiektu domyślnymi wartościami.

Slajd 34



Prototypy



- specjalna właściwość konstruktora
- pozwala na zdefiniowanie elementów wspólnych dla wszystkich obiektów tworzonych za pomocą tego konstruktora
- dostęp
nazwa_funkcji_konstruktora.prototype.nazwa_metody



Na jednym z wcześniejszych slajdów utworzyliśmy prosty obiekt zawierający 2 właściwości i metodę. Jednak definiowanie metod przy każdorazowym tworzeniu obiektu jest niewygodne i nieefektywne. Poznaliśmy też konstruktory obiektu, więc moglibyśmy umieścić definicję metod w konstruktorze, aby nie być zmuszonym do ich ciągłego wpisywania. Takie rozwiązanie mimo, że wygląda dobrze nadal jest nieefektywne. Każda instancja obiektu będzie zawierała własne właściwości oraz definicje metod. Właściwości zwykle są różne dla każdego obiektu danego typu, ale definicje metod są takie same, więc mogłyby być utworzone tylko raz.


W języku JavaScript w tym celu wykorzystuje się tzw. prototyp obiektu. Użycie operatora `new` powoduje nie tylko stworzenie pustego obiektu i przekazanie go funkcji będącej konstruktorem, ale także przygotowanie prototypu obiektu. Wartością prototypu jest wartość właściwości funkcji o nazwie `prototype`. Kiedy do prototypu dodamy nową funkcję lub właściwość będzie ona dostępna dla wszystkich obiektów tworzonych za pomocą tego konstruktora.

Slajd 35



Prototypy cd.

```
function Samochod(producent, model, rocznik) {  
  this.producent = producent;  
  this.model = model;  
  this.rocznik = rocznik;  
}  
function Samochod.prototype.wyswietl = function() {  
  alert(this.producent + " " + this.model);  
}
```



Na jednym z wcześniejszych slajdów utworzyliśmy prosty obiekt zawierający 2 właściwości i metodę. Jednak definiowanie metod przy każdorazowym tworzeniu obiektu jest niewygodne i nieefektywne. Poznaliśmy też konstruktory obiektu, więc moglibyśmy umieścić definicję metod w konstruktorze, aby nie być zmuszonym do ich ciągłego wpisywania. Takie rozwiązanie mimo, że wygląda dobrze nadal jest nieefektywne. Każda instancja obiektu będzie zawierała własne właściwości oraz definicje metod. Właściwości zwykle są różne dla każdego obiektu



Moduł szkoleniowy JavaScript str. 59

Człowiek - najlepsza inwestycja

danego typu, ale definicje metod są takie same, więc mogłyby być utworzone tylko raz.

W języku JavaScript w tym celu wykorzystuje się tzw. prototyp obiektu. Użycie operatora `new` powoduje nie tylko stworzenie pustego obiektu i przekazanie go funkcji będącej konstruktorem, ale także przygotowanie prototypu obiektu. Wartością prototypu jest wartość właściwości funkcji o nazwie `prototype`. Kiedy do prototypu dodamy nową funkcję lub właściwość będzie ona dostępna dla wszystkich obiektów tworzonych za pomocą tego konstruktora.

Slajd 36



 **B2E**
BUSINESS TO EDUCATION  **SZCZECIŃSKI PARK**
NAUKOWO-TECHNOLOGICZNY

Ćwiczenie 1

Napisz kod konstruktora obiektu, będącego reprezentacją koła na płaszczyźnie. W konstruktorze umieść metody zwracające pole i obwód koła.

Napisz kod konstruktora obiektu, będącego reprezentacją koła na płaszczyźnie. W konstruktorze umieść metody zwracające pole i obwód koła.




Slajd 37

 **B2E**
BUSINESS TO EDUCATION  **SZCZECIŃSKI PARK**
NAUKOWO-TECHNOLOGICZNY

Ćwiczenie 1 - rozwiązanie

```
function Kolo(x, y, r) {  
  this.x = x;  
  this.y = y;  
  this.r = r;  
  this.pole = function() {  
    return 3,14 * r * r;  
  }  
  this.obwod = function() {  
    return 2 * 3,14 * r;  
  }  
}
```

Slajd 38


 **B2E**
BUSINESS TO EDUCATION  **SZCZECIŃSKI PARK**
NAUKOWO-TECHNOLOGICZNY
Ćwiczenie 2
Napisz kod konstruktora obiektu reprezentującego prostokąt. W prototypie obiektu umieść metody zwracające pole i obwód prostokąta.


Napisz kod konstruktora obiektu reprezentującego prostokąt. W prototypie obiektu umieść metody zwracające pole i obwód prostokąta.

Slajd 39

 **B2E**
BUSINESS TO EDUCATION  **SZCZECIŃSKI PARK**
NAUKOWO-TECHNOLOGICZNY
Ćwiczenie 2 - rozwiązanie

```
function Prostokat (a, b) {  
  this.a = a;  
  this.b = b;  
}  
Prostokat.prototype.pole = function() {  
  return a * b;  
}  
Prostokat.prototype.obwod = function() {  
  return 2*a + 2*b;  
}
```



9.5.3 Ćwiczenia

Ćwiczenia do lekcji zostały zaprezentowane na slajdach 36 i 38.

Wymagają one od uczestnika umiejętności definiowania obiektów w JavaScript, a także ich konstruktorów, prototypów i funkcji, które mogą być wykorzystane jako metody obiektu.

9.5.4 Opis założonych osiągnięć ucznia

Po ukończeniu tej lekcji uczestnik wie czym są obiekty w języku JavaScript, potrafi je tworzyć. Wie także jak grupować instrukcje w funkcje i w jaki sposób można je wykorzystywać.

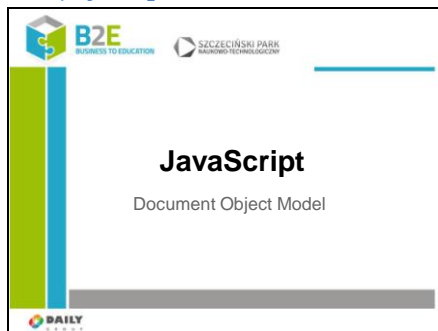
9.6 Lekcja 6 – Model dokumentu – Document Object Model

9.6.1 Cel lekcji

Celem lekcji jest wprowadzenie definicji modelu dokumentu, jako reprezentacji strony internetowej, która może być przetwarzana przez skrypty i programy.

9.6.2 Treść – slajdy z opisem

Slajd 1

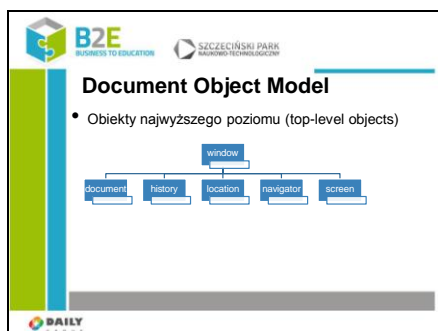


Każda strona www składa się z wielu elementów (linków, paragrafów, obrazów itp.) Każdy z tych obiektów może mieć swoje właściwości (jak rozmiar, kolor) i można wykonywać na nich różne operacje.

Dodatkowo dostępne są jeszcze obiekty opisujące okno przeglądarki i dokumentu załadowanego w bieżącym oknie.

Do wygodnego zarządzania wszystkimi obiektami stworzono model dokumentu – Dokument Object Model, który określa hierarchię obiektów na stronie, ich właściwości oraz sposób zachowania.

Slajd 2



Obsługą standardu zajmuje się organizacja W3C (World Wide Web Consortium – <http://www.w3.org>), ale poszczególne przeglądarki w różnym stopniu respektują zalecenia standardu.

Document Object Model zmienia każdy element witryny www w hierarchie obiektów, z których każdy posiada swoje własne właściwości. Właściwości opisują wszystkie atrybuty obiektu takie jak HTML, który zawiera, jego wysokość, szerokość itd.

Najbardziej zewnętrznym elementem hierarchii (jej korzeniem) jest obiekt window, który jest bieżącym oknem przeglądarki, zakładką, ramką pływającą (iframe) lub oknem typu popup.

Poniżej znajdują się następujące obiekty:



document który jest reprezentacją dokumentu (X)HTML,

obiekt history zawiera historię odwiedzin stron podczas danej sesji przeglądarki,

obiekt location reprezentuje adres URL aktualnego dokumentu,


obiekt navigator przechowuje informacje o przeglądarce, jej nazwę, wersję itp

Slajd 3



Obiekt window

- główny i domyślny obiekt w DOM
- reprezentuje okno lub zakładkę w przeglądarce
- zmienne globalne w JavaScript są w praktyce właściwościami tego obiektu




Obiekt window reprezentuje okno przeglądarki (lub jeśli korzystamy z przeglądarek typu Firefox – jego zakładkę). Jest obiektem domyślnym tzn. że jego metody i właściwości możemy wywoływać bezpośrednio – bez konieczności podawania jego nazwy.

Slajd 4

Przykładowe metody obiektu window

- alert(treść) – wyświetla komunikat zawarty w argumente treść
- confirm(treść) – wyświetla komunikat zawarty w argumente treść oraz przyciski OK i Anuluj
- prompt(treść [opis]) – wyświetla okno dialogowe z polem pozwalającym wprowadzić odpowiedź
- back() – odpowiednik przycisku wstecz w przeglądarce
- close() – zamyka okno
- open(URL, nazwa [właściwości, [zamiana]]) – wyświetla nowe okno o określonej nazwie.



Metoda alert wyświetla okienko z komunikatem i przyciskiem OK, metoda confirm pozwala nam uzyskanie potwierdzenia od użytkownika – wyświetlone zostanie okno z komunikatem i przyciski OK, i Anuluj.


Wynikiem wywołania metody będzie true – jeśli wciśnięto OK lub false jeśli wciśnięto Anuluj.

Metoda prompt wyświetla komunikat zawarty w parametrze treść i pole tekstowe na wprowadzenie odpowiedzi. W polu tekstowym zostanie wyświetlony tekst wprowadzony jako opis, wynikiem działania jest treść wpisana przez użytkownika

Metoda back() cofnie nas o 1 stronę w historii przeglądarki

Metoda `close()` zamyka okno przeglądarki
 Metoda `open()` otwiera adres Url w nowym oknie przeglądarki o podanej nazwie i właściwościach (np. rozmiarze)

Slajd 5



Obiekt document

zawiera dokument html wczytany do przeglądarki
 pozwala na dostęp i manipulacje praktycznie każdym elementem strony

Obiekt `document` reprezentuje dokument html wczytany do okna przeglądarki oraz zawiera szereg właściwości i metod pozwalających na jego modyfikację.
 Poprzez ten obiekt można otrzymać dostęp praktycznie do każdego elementu strony i za jego pomocą można tymi elementami manipulować.

Slajd 6



Podstawowe metody obiektu document

- `write(tekst)`
- `writeln (tekst)`
- `getElementById(id)`
- `getElementsByName(nazwa)`
- `getElementsByTagName(tag)`

- `write(tekst)` – wpisuje w dokumencie tekst zawarty w argumencie tekst
- `writeln (tekst)` – tak samo jak `write`, ale na końcu dokleja znak nowej linii
- `getElementById(id)` – zwraca odnośnik do obiektu o identyfikatorze id
- `getElementsByName(nazwa)` – zwraca listę obiektów o atrybucie nazwa równym przekazanej nazwie
- `getElementsByTagName(tag)` – zwraca listę obiektów o danym znaczniku



Slajd 7



Przechowuje historię stron odwiedzanych przez użytkownika w danej sesji przeglądarki. Zawiera metody pozwalające na przemieszczanie się między odwiedzanymi stronami

`history.current` – zawiera adres URL bieżącej strony,
`history.length` – liczba stron przechowywanych w historii
właściwości `next` i `previous` zawierają adresy stron dostępnych pod przyciskami wstecz i dalej

Podstawowe metody obiektu to:
`back()` i `forward()` – wczytują odpowiednio poprzednią i następną stronę w historii

`go` – wczytuje stronę podaną jako parametr. Jeśli parametr jest liczbą to jest traktowany jako pozycja względem aktualnej strony w historii przeglądania. Jeśli parametr jest ciągiem znaków to traktowany jest jako adres URL

Slajd 8



Zawiera adres URL aktualnego dokumentu oraz metody pozwalające na manipulację tym adresem. We właściwościach obiektu znajdują się informacje składowe adresu (`host`, nazwa hosta, `port`).



Podstawowe metody:

- `assign(url)` – powoduje wczytanie dokumentu o adresie wskazywanym w argumencie URL
- `reload(force)` – wymusza ponowne wczytanie strony. Jeśli parametr `force` jest ustawiony na `true` to strona jest ponownie wczytywana z serwera, w przeciwnym wypadku może zostać wczytana z pamięci cache
- `replace(url)` – powoduje wczytanie dokumentu o adresie wskazywanym w argumencie URL, bieżąca strona nie zostanie

Moduł szkoleniowy JavaScript str. 65


zapamiętana w historii

Slajd 9



 

Obiekt navigator

Zawiera informacje o przeglądarce i systemie operacyjnym na którym działa
Zawiera właściwość userAgent, którą można wykorzystać do określenia typu przeglądarki




Slajd 10



Dostęp do elementów strony

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Tester JavaScript</title>
<script type="text/javascript">
</script>
<body>
<div id="content">
Witaj!
</div>
</body>
</html>
```




Strona www jest wczytywana w przeglądarce jako struktura hierarchiczna (drzewo), której elementy są zbudowane ze znaczników html. Wykorzystując JavaScript możemy wyszukiwać i modyfikować elementy, usuwać i tworzyć nowe.

Slajd 11



Dostęp do elementów strony

```
metoda getElementById()  
var content = document.getElementById("content");  
var tresc = content.innerHTML;  
content.innerHTML = tresc;
```





Strona www jest wczytywana w przeglądarce jako struktura hierarchiczna (drzewo), której elementy są zbudowane ze znaczników html. Wykorzystując JavaScript możemy wyszukiwać i modyfikować elementy, usuwać i tworzyć nowe.

Do znalezienia danego obiektu najlepiej wykorzystać metodę `getElementById` obiektu `document`. Zwraca ona referencje do obiektu w drzewie dokumentu o podanym id. Taką referencję najlepiej przypisać do zmiennej i traktować jako obiekt.


Do modyfikacji elementu można wykorzystać właściwość `innerHTML`. Większość elementów drzewa DOM posiada taką właściwość. `innerHTML` to zawartość obiektu w postaci HTML. Możemy odczytać zawartość wyszukanego przez nas elementu (w naszym przypadku będzie to napis „Witaj!”) lub zmodyfikować

Slajd 12



Bezpośrednia manipulacja węzłami dokumentu

```
każdy element nie będący tekstem zawiera kolekcję  
węzłów potomnych (childNodes)  
var zmienna = content.childNodes[0]  
zmienna.nodeValue = „Nowa treść”;
```





W `innerHTML` możemy umieścić treść będącą kodem HTML, po przypisaniu tej wartości przeglądarka sama przebuduje drzewo DOM dokumentu. Czasami jednak wymagana jest większa kontrola nad węzłami dokumentu.

W DOM każdy węzeł nie będący tekstem zawiera kolekcję węzłów potomnych. Np. węzeł `<body>` zawiera węzeł potomny `<div>`, który zawiera węzeł potomny z tekstem Witaj.


Dostęp do elementów potomnych realizowany jest za pomocą tablicy `childNodes`. W zmiennej `zmienna` uzyskamy 1 element potomny węzła `content`. Dostęp do wartości tego obiektu możliwy jest przez właściwość `nodeValue`

Slajd 13



Tworzenie elementów przez skrypt

Węzły powiązane z elementami html
`var div = document.createElement("div");`
Węzły tekstowe
`var tekst = document.createTextNode("Treść");`
Łączenie:
`div.appendChild(tekst);`
`content.appendChild(div);`




Za pomocą JavaScriptu możemy nie tylko modyfikować istniejące ale także dodawać nowe elementy do strony. Aby tego dokonać trzeba wyróżnić kilka typów węzłów:

węzły powiązane z elementami html tworzymy za pomocą metody `createElement` obiektu `document` – jako argument podajemy nazwę tworzonego znacznika.

Natomiast węzły zawierające tylko tekst tworzymy za pomocą innej metody obiektu `document` - `createTextNode`. Tak utworzone węzły łączymy ze sobą za pomocą metody `appendChild`


Żeby taki połączony węzeł był widoczny musimy połączyć go z innym elementem naszej strony.

Slajd 14



Usuwanie elementów

`element.removeChild(obiekt);`
`np. content.removeChild(div);`



Skoro możemy modyfikować i dodawać elementy to pewnie istnieje też sposób na usuwanie elementów. Służy do tego metoda `removeChild`, której jako parametr przekazujemy obiekt do usunięcia.

`element` jest węzłem z którego usuwamy węzeł potomny.

Usunięcie obiektu jest tylko odłączeniem go od elementu nadrzędnego, a nie usunięciem z pamięci. Węzeł nadal istnieje a referencje do niego otrzymaliśmy w wyniku działania metody `removeChild`.

9.6.3 Ćwiczenia

Napisz skrypt wyszukujący na stronie element o identyfikatorze "test". Zmień kolor jego tła na zielony.

Napisz skrypt umieszczający treść pierwszego paragrafu strony w znaczniku `<h1>`

9.6.4 Opis założonych osiągnięć ucznia

Uczestnik po zakończeniu lekcji posiada podstawowe informacje na temat drzewa dokumentu (modelu DOM). Potrafi wyszukiwać w nim elementy i je modyfikować.

9.7 Lekcja 7 – Zdarzenia

9.7.1 Cel lekcji

Interakcja na stronach www realizowana jest przez zdarzenia. „Coś” się na stronie wydarzyło. Celem lekcji jest wprowadzenie pojęcia zdarzenie i pokazanie w jaki sposób można reagować na zmiany na naszej witrynie.

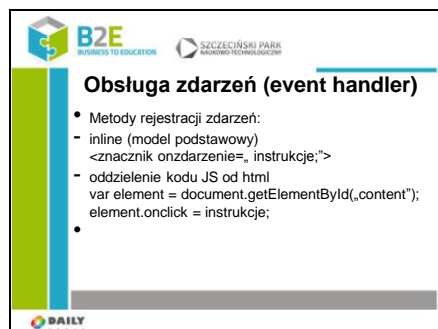
9.7.2 Treść – slajdy z opisem

Slajd 1



Zdarzeniem w JavaScript może być kliknięcie myszą, przesunięcie kursora nad element, opuszczenie strony itp. W JS możemy zdefiniować specjalne funkcje zwane funkcjami obsługi zdarzeń (z ang. event handler). W tej lekcji zobaczymy jakie typy zdarzeń występują oraz jak włączyć ich obsługę.

Slajd 2



Reagowanie na zdarzenia zachodzące na stronie zwiększa jej dynamikę i atrakcyjność. Kiedy coś się dzieje na stronie przeglądarka generuje zdarzenie (obiekt) a następnie bada czy jest zarejestrowana funkcja obsługi tego zdarzenia. Jeśli tak to funkcja ta jest wywoływana.

Istnieje kilka metod przypisywania rejestracji zdarzeń w JavaScriptcie

- inline, zwany też podstawowym – najstarszy, obsługiwany podobnie przez większość przeglądarek.

Zdarzenie zwykle utożsamiane z nazwą funkcji obsługującej, która jest jednocześnie nazwą atrybutu znacznika html, do którego podpinamy zdarzenie. Ten sposób ma kilka wad

- po pierwsze miesza kod JavaScript z treścią strony,
- nie przekazuje automatycznej referencji do obiektu, który wywołał zdarzenie (można to obejść

Moduł szkoleniowy JavaScript str. 69

Człowiek - najlepsza inwestycja

przekazując referencje `this` jako argument wywołania funkcji)

- oddzielenie kodu JS od html – umieszczenie wielu instrukcji JavaScript bezpośrednio w znaczniku jest możliwe, ale jeśli chcemy wykorzystać nasz skrypt ponownie to musimy skopiować jego treść i podłączyć do nowego elementu. Możemy to uprościć i przenieść nasz kod do funkcji a w znacznikach umieścić tylko wywołania naszych funkcji. Jednak gdy będziemy chcieli zmienić obsługę części zdarzeń na naszej stronie będziemy musieli przeszukiwać dokument html i wyszukiwać elementy, które wywołują naszą funkcję. Dlatego możliwe jest podłączenie obsługi zdarzenia bezpośrednio w kodzie skryptu. Możemy do tego wykorzystać poznaną wcześniej metodę `getElementById`, aby zlokalizować dany element na stronie i umieścić w nim obsługę zdarzenia. Aby usunąć obsługę zdarzenia przypisujemy mu wartość `null`.

Slajd 3

 
Obsługa zdarzeń (event handler)

- Metody rejestracji zdarzeń:
 - nowy model rejestracji zdarzeń (`addEventListener`)
- `var element = document.getElementById('przycisk');`
 - `element.addEventListener('click', przemies, false);`
 - `element.addEventListener('click', wypiszTekst, false);`
 - `element.addEventListener('click', function() {`
 - `this.style.color = 'blue';`
 - `}, false);`
 - `- removeEventListener()`
 - `element.removeEventListener('click', przemies, false);`



Wadą wcześniej przedstawionych sposobów jest możliwość podłączenia tylko jednej funkcji pod dane zdarzenie. Jeśli w trakcie działania przypiszemy do zdarzenia nową funkcję to nadpiszemy starszą.

Możemy to obejść używając „nowego” modelu rejestracji zdarzeń z wykorzystaniem metody `addEventListener()`;

Przyjmuje ona 3 parametry: typ zdarzenia, funkcja do wywołania parametr włączający lub wyłączający bąbelkowe wywołania zdarzeń (z reguły jest on ustawiony na wartość `false`)

Bąbelkowe wywołanie zdarzeń:

(jeśli jest ustawione na `true`, zdarzenie jest wywoływane dla wszystkich

Moduł szkoleniowy JavaScript str. 70

Człowiek - najlepsza inwestycja

elementów nadrzędnych – czyli jeśli klikniemy w obszarze div, który jest częścią innego obszaru to zdarzenie onClick zostanie przekazane do obu elementów!. Jeśli oba elementy zawierają funkcję obsługującą to zdarzenie to obie funkcje zostaną wywołane).

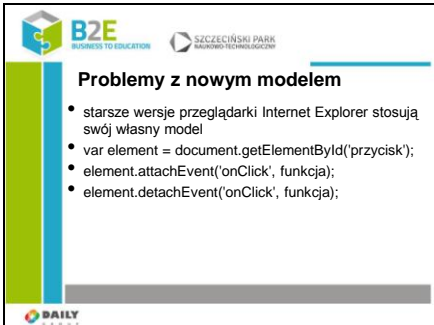
Zdarzenie „wędruje” od elementu który je wywołał w górę dokumentu (aż do elementu document) tak jak bąbelki powietrza wędrują w górę butelki.

W elemencie przycisk przypisujemy 3 funkcje obsługujące kliknięcie: przenies, wypiszTekst, i anonimową funkcję zmieniającą kolor na niebieski. Od tego momentu kliknięcie na elemencie uruchomi wszystkie 3 funkcje.

Aby usunąć obsługę zdarzenia z elementu wykorzystujemy metodę `removeEventListener()`, której jako argument podajemy takie same parametry jak metodzie `addEventListener`.

Niestety nie możemy jednak usunąć anonimowych funkcji!

Slajd 4




Problemy z nowym modelem

- starsze wersje przeglądarki Internet Explorer stosują swój własny model
- `var element = document.getElementById('przycisk');`
- `element.attachEvent('onClick', funkcja);`
- `element.detachEvent('onClick', funkcja);`

Wcześniejsze wersje Internet Explorera (≤ 8) stosują własny model. Zamiast zdarzenia `addEventListener` i `removeEventListener` są zdarzenia `attachEvent` i `detachEvent`. Jako parametry tych funkcji podajemy nazwę zdarzenia z przedrostkiem `on` np. `onClick` a jako 2 nazwę funkcji obsługującej zdarzenie.



Slajd 5



Słowo kluczowe this

JavaScript udostępnia słowo kluczowe this, które jest referencją do obiektu, wywołującego metodę.



```
function zmienKolor(kolor) {  
  this.style.color = kolor;  
}  
element.zmienKolor('#808080');
```



JavaScript udostępnia słowo kluczowe this, które jest referencją do obiektu, wywołującego metodę.

Przykładowo jeśli chcemy stworzyć funkcję która zmieni kolor elementu możemy napisać ją w następujący sposób i wywołać

Slajd 6



Dodatkowe opcje


- blokowanie domyślnej akcji (preventDefault())

```
element.addEventListener('click',function (e) {  
  alert('Ten link nigdzie nie przeniesie.');
```

```
  e.preventDefault();  
},false)
```
- blokowanie nasłuchu innych zdarzeń (stopPropagation())

```
a.addEventListener('click',function (e) {  
  alert('Kliknięto link');
```

```
  e.preventDefault();  
  e.stopPropagation();  
},false);
```



Elementy na stronie mają w większości przypadków przypisane domyślne akcje. Np. link przenosi w inne miejsce, formularz jest wysyłany na serwer itp.

Dodane przez nas zdarzenia zostaną wykonane jako pierwsze, ale później zostanie wykonana domyślna czynność.

Aby temu zapobiec możemy skorzystać z metody e.preventDefault(), gdzie e to obiekt zdarzenia.

Jak wspominałem wcześniej zdarzenie po wystąpieniu przekazywane jest w górę dokumentu – aż do znacznika document i jeśli po drodze napotka na metodę obsługującą dane zdarzenie to metoda ta zostanie wywołana.

Aby wstrzymać takie działanie możemy wykorzystać metodę stopPropagation()



Slajd 7



Lista typowych zdarzeń obsługiwanych przez przeglądarkę:

- onblur: zdarzenie uruchamiane gdy element straci focus (zmienimy aktywny element na inny)
- onchange: element traci focus i zmieniła się zawartość elementu
- onclick: zdarzenie, które powstaje kiedy element został kliknięty
- ondblclick:
- onfocus: element otrzymuje fokus (jest aktywny)
- onkeypress: zdarzenie powstaje kiedy klawisz zostaje wciśnięty i puszczony (są jeszcze zdarzenia onkeydown i onkeyup – wywoływane po naciśnięciu i puszczeniu przycisku)
- onload: przeglądarka zakończyła ładowanie strony lub ramki

Slajd 8



Lista typowych zdarzeń obsługiwanych przez przeglądarkę:

- onmousedown: zdarzenie powstaje gdy klawisz myszy zostanie naciśnięty nad elementem
- onmousemove: klawisz myszy jest przesuwany nad elementem
- onmouseout: kursor myszy opuścił element
- onmouseover: kursor myszy wszedł w element
- onmouseup: przycisk myszy został zwolniony nad elementem
- onresize: zmienił się rozmiar okna



Slajd 9

Ładowanie strony

```
<html>
<head>
<title>Moja strona</title>
</head>
<body onload="alert('załadowano stronę');">
</body>
</html>
```

Komunikat „Załadowano stronę” pojawi się dopiero po wczytaniu całej strony do przeglądarki. Jest to bardzo przydatne, jeśli nasza funkcja ma działać na elementach strony. W tym przypadku mamy pewność, że wszystkie elementy są już wczytane i utworzone w przeglądarce a nasza metoda będzie mogła z nich skorzystać. Gdybyśmy umieścili wywołanie naszej metody poza zdarzeniem zostałaby wywołana w momencie wczytania przez przeglądarkę, bez oczekiwania na zakończenie wczytywania strony.

Slajd 10

Obsługa kliknięć

- onclick
<div id="przycisk" style="background-color: red; width:100px; height: 100px;" onclick="alert('Witaj!');">
</div>
- ondblclick
<div id="przycisk" style="background-color: red; width:100px; height: 100px;" ondblclick="alert('Witaj 2x!')">
</div>

Najczęściej wykorzystywane zdarzenie. Metodę obsługi zdarzenia możemy przypisać do większości elementów strony.

W tym przykładzie utworzyliśmy element div o rozmiarze 100 na 100px i podłączyliśmy do niego zdarzenie click z instrukcją wywołującą okno z komunikatem.

Zdarzenie dblclick jest wywoływane po dwukrotnym kliknięciu na element. Jeśli element ma podłączoną metodę onclick to zostanie ona wywołana zamiast oczekiwanej przez nas metody ondblclick

Slajd 11

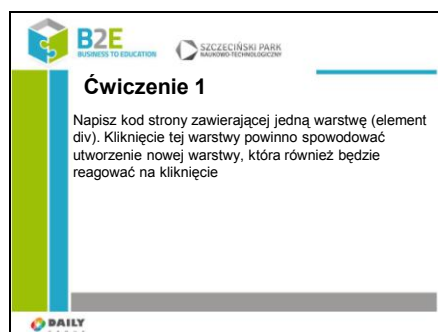
Reakcja na ruch myszy

zdarzenia onmouseover i onmouseout

```
<div id="przycisk" style="background-color: red; width:100px; height: 100px;" onmouseover="alert('kursor nad elementem');">  
</div>  
<div id="przycisk" style="background-color: red; width:100px; height: 100px;" onmouseout="alert('kursor opuścił element');">  
</div>
```

zdarzenia onmouseover i onmouseout służą do wykrywania czy kursor myszy znalazł się lub opuścił dany element html.

Slajd 12



Ćwiczenie 1

Napisz kod strony zawierającej jedną warstwę (element div). Kliknięcie tej warstwy powinno spowodować utworzenie nowej warstwy, która również będzie reagować na kliknięcie

Napisz kod strony zawierającej jedną warstwę (element div). Kliknięcie tej warstwy powinno spowodować utworzenie nowej warstwy, która również będzie reagować na kliknięcie

9.7.3 Ćwiczenia

Ćwiczenie zostało zaprezentowane na slajdzie 12. Wymaga ono od uczestnika opanowania sposobu definiowania obsługi zdarzeń i tworzenia nowych obiektów na stronie

9.7.4 Opis założonych osiągnięć ucznia

Uczestnik potrafi wymienić podstawowe zdarzenia na stronie, potrafi napisać funkcję obsługującą dowolne z tych zdarzeń.

9.8 Lekcja 8 – Obsługa błędów

9.8.1 Cel lekcji

Celem lekcji jest przedstawienie mechanizmów obsługi błędów programu w języku JavaScript. Zaprezentowane zostaną podstawowe instrukcje służące do zapewnienia poprawnego wykonania programu.

9.8.2 Treść – slajdy z opisem

Slajd 1



JavaScript

Obsługa błędów

W każdym programie występują błędy. Nawet najlepsi programiści popełniają błędy. W języku JavaScript oprócz typowych błędów popełnianych przez programistę, możliwe są także błędy związane z różną interpretacją standardu ECMAScript w przeglądarkach internetowych. Np. funkcja, która zadziała w przeglądarce Google Chrome, niekoniecznie musi działać tak samo w przeglądarce Firefox.

Do obsługi błędów w języku JavaScript wykorzystywane są przeniesione z języka C++ instrukcje try catch, oraz obiekty typu Error.

Moduł szkoleniowy JavaScript str. 75

Człowiek - najlepsza inwestycja

Program może wskazać
nieprawidłowe działanie przez
zgłoszenie (rzucenie) wyjątku.

Slajd 2



Wyjątki (ang. exceptions) to konstrukcje służące do obsługi błędów. Obsługują one sytuacje, w których program nie zadziałał tak jak założył programista. Np. użytkownik przekazał tekst w polu przeznaczonym na liczbę; nie zadziałało pobranie danych itp.

Wyjątki są informacją od programu, że dana instrukcja się nie powiodła. Część wyjątków zgłaszana jest przez funkcje systemowe (np. konwersje typów, próby odczytania nieistniejącej właściwości itp.), ale nic nie stoi na przeszkodzie aby zgłaszać własne typy wyjątków w tworzonych przez nas funkcjach i metodach. Służy do tego instrukcja throw, której postać to: throw wyrażenie. Spowoduje to wypisanie na konsoli przeglądarki tekstu zawartego w wyrażeniu.



Slajd 3

Zdarzenie onerror

- Najprostszą metodą obsługi błędu to podłączenie funkcji do zdarzenia onError

```
window.onerror = function(komunikat, url, linia) {  
    instrukcje;  
}
```

Najprostszą metodą obsługi błędu to podłączenie funkcji do zdarzenia onError. Zdarzenie onError przekazuje 3 parametry do funkcji obsługującej: komunikat o błędzie, adres url strony oraz numer linii w której wystąpił błąd.

Slajd 4



Obsługa błędów

- Obiekt Error
- throw new Error();
- throw new Error("Wystąpił błąd!");

Zamiast jednak stosować typy proste przy tworzeniu wyjątków należy skorzystać z obiektu opisującego błąd. W języku JavaScript jest to obiekt Error. Jako parametr wywołania konstruktora możemy podać treść komunikatu o błędzie. Utworzony obiekt ma 2 główne właściwości: name (to nazwa konstruktora wykorzystanego do utworzenia obiektu) oraz message – treść komunikatu podana w konstruktorze. Obiekt Error zawiera metodę toString(), która może się różnić w zależności od przeglądarki – Mozilla Firefox wyświetla domyślnie właściwości name i message oddzielone dwukropkiem.




Slajd 5



Obsługa błędów

Instrukcje do przechwytywania błędów:

- try ... catch
- try ... catch ... finally
- zdarzenie onerror



Wcześniejsze przykłady pokazywały sposoby zgłaszania wyjątków, które nie były w żaden sposób obsługiwane w naszym skrypcie. Wynikiem było więc wypisanie wyjątku na konsoli przeglądarki. Zamiast wypisywać komunikaty o błędach należy odpowiednio zareagować w kodzie naszego programu. W języku JavaScript do obsługi wyjątków wykorzystywane są instrukcje zapożyczone z innych języków programowania takich jak C++ i Java. Są to instrukcje try catch, try catch z dodatkową klauzulą finally, oraz zdarzenie onerror

Slajd 6



Instrukcja try ... catch

- Instrukcja try ... catch ma następującą postać
- try {
- instrukcje;
- }
- catch (wyjątek) {
- obsługa wyjątku
- }



Instrukcja try catch ma następującą postać:

w bloku try umieszczamy kod, który chcemy wykonać i w którym może wystąpić błąd, następnie umieszczamy instrukcję catch z parametrem, w którym zostanie przekazany obiekt naszego wyjątku. Dzięki temu będziemy mieć dostęp do typu wyjątku, i komunikatu, który został w nim zdefiniowany

Slajd 7



Instrukcja finally



- Instrukcje try catch można rozszerzyć o opcjonalną instrukcję finally.
- try {
- instrukcje;
- }
- catch (wyjątek) {
- kod obsługi wyjątku;
- }
- finally {
- instrukcje;
- }



Instrukcje try catch możemy rozszerzyć o opcjonalną instrukcję finally. Dodanie tej klauzuli spowoduje że instrukcje wpisane w tej klauzuli zostaną wykonane zawsze, nawet gdy zostanie zgłoszony wyjątek.




Slajd 8





Zagnieżdżanie instrukcji try catch

```
try{
  //instrukcje, które mogą spowodować wyjątek 1
  try{
    //instrukcje, które mogą spowodować wyjątek 2
  }
  catch (wyjątek2){
    //obsługa wyjątku 2
  }
}
catch (wyjątek1){
  //obsługa 1 wyjątku
}
```




Instrukcje try catch można zagnieżdżać. Tzn, wewnątrz instrukcji try możemy umieścić kolejny blok obsługujący inny wyjątek naszego programu.

Slajd 9





Propagacja wyjątków

- W przypadku wystąpienia wyjątku jest on przekazywany do najbliższego bloku obsługi błędów. Jeśli funkcja nie zawiera obsługi błędów to wyjątek przekazywany jest do miejsca wywołania funkcji.
- Po obsłużeniu wyjątku działanie programu jest wznowiane.
- Jeśli program nie zawiera obsługi błędów, działanie całego skryptu jest zatrzymywane i informacja o błędzie wpisywana jest na konsoli przeglądarki




W momencie wystąpienia wyjątku wstrzymywane jest wykonywanie kodu skryptu, a sterowanie zostanie przekazane do najbliższego bloku obsługi wyjątku. Jeśli definicja funkcji nie zawiera obsługi błędów wyjątek jest przekazywany do miejsca wywołania funkcji. Po obsłużeniu błędu wykonywanie programu jest kontynuowane, a jeśli w całym skrypcie nie znajdzie się blok obsługi wyjątku to działanie skryptu zostanie przerwane i komunikat o błędzie będzie wypisany na konsoli.

Slajd 10



Predefiniowane obiekty wyjątków

- Oprócz typu Error w trzeciej wersji specyfikacji ECMAScript wprowadzono dodatkowe typy błędów:
- EvalError
- RangeError
- ReferenceError
- SyntaxError
- TypeError
- URIError



Oprócz typu Error ECMAScript v3 definiuje jeszcze 6 innych: EvalError, RangeError, ReferenceError, SyntaxError, TypeError i URIError. Są dostępne od wersji 1.5 JavaScriptu. Sposób tworzenia obiektów tych typów jest taki sam jak obiektu Error.

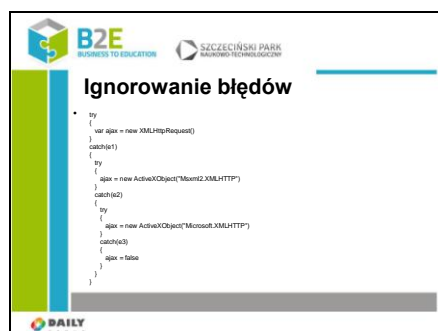
- EvalError jest wykorzystywany w przypadku nieprawidłowego użycia funkcji eval.
- RangeError jest wykorzystywany , gdy wartość numeryczna przekracza dopuszczalny zakres.
- ReferenceError jest wykorzystywany przy próbie odczytu nieistniejących zmiennych.
- SyntaxError jest wykorzystywany po

wykryciu błędu składniowego, np. przez metodę eval oraz konstruktory Function i RegExp.

- TypeError jest wykorzystywany, gdy wartość jest typu innego niż oczekiwany. Może tak być przy próbie dostępu do właściwości o wartości null bądź undefined, użycia operatora new i argumentu niebędącego konstruktorem, próbie wywołania nieistniejącej metody obiektu itp.

- URIError jest wykorzystywany przez metody kodujące bądź dekodujące adresy URI, gdy wykryte zostaną nieprawidłowo sformowane dane.

Slajd 11

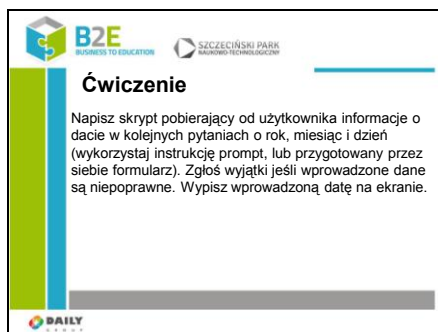


Slajd 11: Ignorowanie błędów. Zawiera logo B2E, Szczeciński Park Naukowo-Technologiczny oraz DAILY GROUP. Tytuł: Ignorowanie błędów. Kod JavaScript:

```
try {
    var ajax = new XMLHttpRequest()
} catch(e1) {
    try {
        ajax = new ActiveXObject("Msxml2.XMLHTTP")
    } catch(e2) {
        try {
            ajax = new ActiveXObject("Microsoft.XMLHTTP")
        } catch(e3) {
            ajax = false
        }
    }
}
```

W klauzuli catch nie musimy tylko informować użytkownika o napotkanym błędzie, tylko dodać np. inne instrukcje i kontynuować wykonanie programu. Jest to mechanizm przydatny przy tworzeniu wersji skryptu działającej w wielu przeglądarkach. W przykładzie pokazano utworzenie obiektu AJAX dla różnych przeglądarek. Jeśli korzystamy z nowszych przeglądarek wykonana zostanie instrukcja pierwsza, natomiast jeśli korzystamy z przeglądarki internet explorer 7 to 1 instrukcja zgłosi błąd i w obsłudze błędu spróbuje wykonać instrukcję tworzącą obiekt żądania za pomocą metody specyficznej dla internet explorera.

Slajd 12



Ćwiczenie

Napisz skrypt pobierający od użytkownika informacje o dacie w kolejnych pytaniach o rok, miesiąc i dzień (wykorzystaj instrukcję prompt, lub przygotowany przez siebie formularz). Zgłoś wyjątki jeśli wprowadzone dane są niepoprawne. Wypisz wprowadzoną datę na ekranie.

Napisz skrypt pobierający od użytkownika informacje o dacie (wykorzystaj instrukcję prompt, lub przygotowany przez siebie formularz). Zgłoś wyjątki jeśli wprowadzone dane są niepoprawne.

Wyjątki powinny być zgłoszone w przypadku podanie nieprawidłowej cyfry (w przypadku miesiąca spoza zakresu 1-12 itd.), w przypadku wpisania w pole tekstu.

Jeśli data zostanie wprowadzona poprawnie wypisz ją na ekranie.

9.8.3 Ćwiczenie

Napisz skrypt pobierający od użytkownika informacje o dacie (wykorzystaj instrukcję prompt, lub przygotowany przez siebie formularz). Zgłoś wyjątki jeśli wprowadzone dane są niepoprawne.

Wyjątki powinny być zgłoszone w przypadku podanie nieprawidłowej cyfry (w przypadku miesiąca spoza zakresu 1-12 itd.), w przypadku wpisania w pole tekstu.

Jeśli data zostanie wprowadzona poprawnie wypisz ją na ekranie.

Ćwiczenie obrazuje praktyczny sposób wykorzystania instrukcji poznanych w czasie lekcji.

9.8.4 Opis założonych osiągnięć ucznia

Uczestnik potrafi dodać do skryptu metody obsługi błędów.

9.9 Lekcja 9 – Wykorzystanie języka JavaScript

9.9.1 Cel lekcji

Po wprowadzeniu podstawowych elementów języka JavaScript czas pokazać do czego można ten język wykorzystać. W lekcji zostaną przedstawione podstawowe zastosowania języka JavaScript – takie jak walidacja przesyłanych formularzy, modyfikacja elementów css, operacje na łańcuchach znaków.

9.9.2 Treść – slajdy z opisem

Slajd 1



W lekcji omówimy najczęstsze sposoby wykorzystania języka JavaScript.

- zmiana elementów css
- operacje na łańcuchach znaków
- wykorzystanie do walidacji formularzy

Zobaczymy jak wykorzystać zaprezentowane na wcześniejszych lekcjach elementy języka JavaScript

Slajd 2



Style CSS to ważny element każdej nowoczesnej witryny WWW. Za pomocą JavaScriptu możliwa jest dynamiczna zmiana właściwości arkuszy stylów. Style CSS danego elementu przechowywane są w atrybucie „style” danego elementu. Dostęp do tego atrybutu możliwy jest z wykorzystaniem metod `getAttribute`, `setAttribute` i `removeAttribute`

Metoda `getAttribute(„style”)` pobiera wartość atrybutu `style` i umieszcza w zmiennej `styl`.

Aby ustawić styl dla obiektu wywołujemy metodę `setAttribute` z dwoma parametrami. Pierwszym jest nazwa atrybutu (w naszym przypadku `style`) a drugim jego wartość.

Usunięcie stylu możliwe jest za pomocą metody `removeAttribute`.

Slajd 3

Obiekt style
każdy obiekt reprezentujący element strony zawiera właściwość style.
Jest ona odwzorowaniem atrybutu style
 obekt.style.

```
<div style="background-color: blue; width: 100px; height: 100px;">  
element.style.backgroundColor = "blue";  
element.style.width = "100px";  
element.style.height = "100px";  
</div>
```

Wykorzystanie atrybutu style jest wygodne jeśli chcemy zmodyfikować całą jego zawartość. Jeśli jednak chcemy zmienić tylko jedną właściwość (np, kolor tła) pobranie całego atrybutu będzie wymagało od nas obsłużenia także innych właściwości, ustawionych dla tego elementu. Będziemy musieli wyszukać w atrybucie aktualny kolor tła, a przy zapisie zmodyfikowanej wartości pamiętać także o pozostałych ustawionych właściwościach.

Na szczęście w JavaScript każdy element ma swój obiekt style (który jest odwzorowaniem atrybutu style).

Jeśli w zmiennej obiekt przechowujemy referencje do elementu witryny do dostęp do obiektu style możliwy jest przez wypisanie obiekt.style.

Atrybuty css danego elementu są odwzorowywane do właściwości obiektu style w następujący sposób: jeśli atrybut elementu nazywał się background-color to jego reprezentacja w JavaScript będzie wyglądała następująco: backgroundColor

Slajd 4

Właściwość className

- odzwierciedla atrybut class elementu html
- ```
<div id="przycisk" class="niebieski"></div>
```
- ```
var element = document.getElementById("przycisk");  
element.className = "zielony";
```



Do tej pory zmienialiśmy tylko konkretne atrybuty stylu przypisanego do danego elementu. Stosując JavaScript możemy również zmienić wartość atrybutu class elementu html, a co za tym idzie zmienić cały styl przypisany do tego elementu.

Atrybut class zamieniany jest na właściwość className obiektu odpowiadającego danemu znacznikowi.

Aby zmodyfikować właściwość class elementu należy wyszukać go za pomocą metody document.getElementById() a następnie do jego właściwości className przypisać nową wartość.




Slajd 5





Właściwość className

```
<html><head><style type="text/css">
.zielony { background-color: green; }
.niebieski { background-color: blue; }
</style>
<script type="text/javascript">
function zmienKlase() {
var element = document.getElementById("przycisk");
element.className="niebieski"; }
</script></head><body>
<div id="przycisk" class="zielony" style="width: 100px; height: 100px;"><div>
<input type="button" value="zmień" onclick="zmienKlase();"></input>
</div></body></html>
```




Aby zmodyfikować właściwość class elementu należy wyszukać go za pomocą metody `document.getElementById()` a następnie do jego właściwości `className` przypisać nową wartość.

Slajd 6



Ćwiczenie 1


Umieść na stronie przycisk. Niech po kliknięciu zmienia się jego kolor tła i kolor tekstu.
Wykorzystaj obie metody (modyfikacje obiektu style) i zmianę klasy css



Ćwiczenie

Umieść na stronie przycisk. Niech po kliknięciu zmienia się jego kolor tła i kolor tekstu.
Wykorzystaj obie metody (modyfikacje obiektu style) i zmianę klasy css

Slajd 7



Przetwarzanie tekstów



- długość tekstu
- przetwarzanie ciągów



W czasie pisania programów wielokrotnie będziemy potrzebowali wywołać jakieś operacje na ciągach znaków, sprawdzić ich długość, sprawdzić czy zawiera określony ciąg znaków, albo zbadać jaka jest dowolna litera w łańcuchu.
Do przetwarzania tekstów można wykorzystać różne dostarczone w języku metody.



Slajd 8



Jak sprawdzić długość tekstu?

każdy łańcuch znaków w JavaScript jest obiektem
obiekt string zawiera m.in. właściwość length – liczbę
zawartych w nim znaków

```
var tekst = "Witaj świecie!";  
var dlugosc = tekst.length;  
alert("Długość tekstu to: "+dlugosc);
```





każdy łańcuch znaków w JavaScript jest obiektem

obiekt string zawiera m.in. właściwość length – liczbę zawartych w nim znaków.


Powyższy przykład po uruchomieniu pokaże komunikat: "Długość tekstu to 14".

Slajd 9



Przetwarzanie ciągów

- charAt	tekst.charAt(indeks)
- charCodeAt	tekst.charCodeAt(indeks)
- concat	tekst.concat(tekst2...tekstN)
- fromCharCode	String.fromCharCode(kod1..kodN)
- indexOf	tekst.indexOf(wartość [, indeks])
- lastIndexOf	tekst.lastIndexOf(wartość [, indeks])
- match	tekst.match(wyrażenieReg)
- replace	tekst.replace(wyrażenieReg, nowy)



charAt(indeks) – zwraca znak znajdujący się pod wybranym indeksem

charCodeAt(indeks) - zwraca kod znaku znajdującego się pod wybranym indeksem (w JS 1.2 – kod ISO Latin1 w późniejszych Unicode)

concat – łączy łańcuchy znakowe. Wynikiem jest łańcuch powstały z połączenia tekstu oryginalnego i wszystkich tekstów podanych jako parametry

fromCharCode – zwraca łańcuch znaków złożony z podanych jako parametry kodów.

indexOf – zwraca indeks wystąpienia łańcucha wartość w łańcuchu tekst. Jeśli podamy parametr indeks to przeszukiwanie rozpoczyna się od tego indeksu.

lastIndexOf – zwraca indeks ostatniego wystąpienia łańcucha wartość w łańcuchu tekst.

match – zwraca część ciągu tekst pasującego do wyrażenia regularnego, jeśli nie znajdzie dopasowania zwraca null

replace – zwraca ciąg znaków, w którym fragmenty opisane przez wyrażenie zostały zamienione na nowy tekst



Slajd 10

		Przetwarzanie ciągów
- search	tekst.search(wyrażenieReg)	
- slice	tekst.slice(start [, koniec])	
- split	tekst.split([separator [, limit]])	
- substr	tekst.substr(indeks [, liczba])	
- substring	tekst.substring(od, do)	
- toLowerCase	tekst.toLowerCase()	
- toUpperCase	tekst.toUpperCase()	

search – Sprawdza czy ciąg opisany przez wyrażenie występuje w tekście. Jeśli tak to zwracany jest indeks miejsca wystąpienia a w przeciwnym wypadku wartość -1

slice – zwraca podciąg tekstu od indeksu start do końca lub jeśli podaliśmy drugi parametr do wartości drugiego parametru.

split – dzieli ciąg znaków na podciągi względem parametru separator. Jeśli nie podamy separatora to zwracany jest cały ciąg. Parametr limit oznacza maksymalną liczbę zwróconych elementów

substr – zwraca podciąg tekstu, zaczynający się od pozycji oznaczonej przez indeks. Jeśli podamy parametr liczba – zostanie zwrócona taka liczba znaków, jeśli nie podamy tego parametru zostaną zwrócone wszystkie znaki do końca łańcucha.

substring – zostanie zwrócony podciąg rozpoczynający się na pozycji od i kończący na indeksie o pozycji do.

toLowerCase - wszystkie litery w tekście zostaną zmienione na małe.



toUpperCase – wszystkie litery w tekście zostaną zamienione na wielkie.

Slajd 11

		Przetwarzanie ciągów - przykłady
- concat	var t1 = "test".concat("java", "script"); "testjavascript"	
- indexOf	var indeks = "testjavascript".indexOf("va"); 6	
- match	var tekst = "testjavascript".match("java"); java	




Slajd 12



Przetwarzanie ciągów - przykłady



- replace
var tekst = "Witaj %IMIE%!";
"Witaj Tomek!"
- split
var tablica = "a,b,c,d".split(",");
[a,b,c,d]



replace - Metoda zamienia wystąpienie zgodne z wyrażeniem regularnym podanym jako 1 argument na tekst podany jako 2 argument.


split - dzieli tekst na fragmenty względem separatora podanego jako argument, opcjonalnie można podać maksymalną liczbę dopasowanych elementów

Slajd 13



Ćwiczenie

Napisz funkcję javascript sprawdzającą poprawność wpisanego w formularzu adresu e-mail



Ćwiczenie:



Napisz funkcję javascript sprawdzającą poprawność wpisanego w formularzu adresu e-mail

Jako poprawny adres email uznajemy adres:

- zawiera tylko 1 znak @
- znak @ nie jest pierwszym elementem ciągu
- po znaku @ występuje cyfra lub litera.
- w ciągu po @ występuje znak . ale nie jest na ostatniej pozycji

Do napisania algorytmu można wykorzystać następujące metody (indexOf, lastIndexOf, match, search)

Slajd 14




Wprowadzanie danych

- formularze

```
<form name="formularz" action="submit.php" method="POST">  
.....  
<input type="submit" value="wyślij" />  
</form>
```
- sprawdzanie poprawności danych

```
<form name="formularz" action="submit.php"  
onsubmit="return sprawdz();" method="POST">  
.....  
<input type="submit" value="wyślij" />  
</form>
```



Formularz to element strony, który tworzymy za pomocą znacznika form. Dostęp do formularzy możliwy jest przez tablicę forms obiektu document. Formularz służy do przesłania zestawu danych od użytkownika do serwera. Warto wykorzystać JavaScript aby dokonać wstępnej analizy danych wprowadzonych do formularza. Sprawdzenie formularza powinno odbywać się również po stronie serwera, ale nie ma sensu niepotrzebnie obciążać serwera jeśli dane w formularzu są błędnie wprowadzone.



Zobaczmy w jaki sposób możemy wykorzystać napisaną przed chwilą funkcję.

Możemy sprawdzić formularz na dwa sposoby.

Po pierwsze możemy nie umieszczać w nim przycisku typu submit tylko zwykły przycisk typu button i w funkcji obsługującej zdarzenie onclick umieścić sprawdzenie danych i wysyłanie formularza jeśli dane są poprawne.


Nasz przykładowy formularz zawiera 2 pola tekstowe do wprowadzenia nazwy użytkownika i hasła oraz przycisk "Zaloguj". Po kliknięciu przycisku sprawdzane jest czy pola login i password zostały wypełnione (są różne od znaku pustego) i jeśli tak to formularz jest wysyłany przez wywołanie metody submit(). Jeśli którekolwiek z pól jest puste, wyświetlony jest komunikat i działanie funkcji zostanie przerwane.

Slajd 15



Sprawdzanie poprawności danych

```
<form name="logowanie" action="login.php" method="POST">  
  <input type="text" name="login">  
  <input type="password" name="password">  
  <input type="button" value="Zaloguj" onclick="sprawdz()" />  
</form>  
  
function sprawdz() {  
  var loginForm = document.forms["logowanie"];  
  if (loginForm.login == "" || loginForm.password == "") {  
    alert("Nie podano nazwy użytkownika lub hasła");  
    return;  
  }  
  loginForm.submit();  
}
```



Slajd 16

Sprawdzanie poprawności danych

```
<form name="logowanie" action="login.php" method="POST"
onsubmit="return sprawdz();" >
  <input type="text" name="login">
  <input type="password" name="password">
  <input type="submit" value="Zaloguj" >
</form>

function sprawdz() {
  var loginForm = document.forms["logowanie"];
  if (loginForm.login == "" || loginForm.password == "") {
    alert("Nie podano nazwy użytkownika lub hasła");
    return false;
  }
  return true;
}
```

Możemy sprawdzić formularz na dwa sposoby.

Drugim sposobem jest podłączenie obsługi zdarzenia submit do formularza. Ogólna postać jest funkcji jest bardzo podobna. W obsłudze zdarzenia submit wpisaliśmy wywołanie return sprawdz(). Formularz zostanie przesłany do serwera tylko w przypadku gdy funkcja sprawdz zwróci jako wynik true. Stanie się tak, jeśli pola login i password zostaną wypełnione.

Slajd 17

Sprawdzenie poprawności danych

Sprawdzenie poprawności danych można wywołać nie tylko przy wysyłaniu formularza, ale nawet po wprowadzeniu zmian w pojedynczym polu formularza. Służy do tego zdarzenie onchange(). Wykorzystując to zdarzenie a także możliwość wywołania zdarzenia focus() możemy wymusić na użytkowniku wprowadzenie poprawnych wartości do pola tekstowego.

Slajd 18

Przykład

```
<form name="logowanie" >
  <input type="text" name="pesel" onchange="sprawdzPesel();" >
</form>

function sprawdzPesel() {
  var form1 = document.forms["logowanie"];
  if (form1.pesel.value.length != 11) {
    alert("Nieprawidłowy numer PESEL");
    form1.pesel.focus();
  }
}
```

Jeśli wpisujemy liczbę znaków różną od 11 pojawi się komunikat i pole pesel zostanie ustawione jako aktywne.



Co się jednak stanie gdy ponownie opuścimy pole formularza (bez dokonywania poprawek)?

Q: Jak możemy temu zapobiec?

A: wykorzystując zdarzenie onBlur




Slajd 19



Wyrażenia regularne

- wzorce opisujące łańcuchy tekstów
- pozwalają na określenie czy podany ciąg znaków pasuje do wzorca.
- mogą wyszukiwać w tekście wystąpienia wzorca

W JavaScript reprezentowane przez obiekt RegExp




Wyrażenia regularne to wzorce opisujące łańcuchy tekstów. Wyrażenia regularne mogą określać zbiór pasujących łańcuchów albo wskazywać istotne części łańcucha znaków.

Istnieją algorytmy pozwalające sprawdzić czy podany ciąg znaków pasuje do wzorca.

Za pomocą wyrażeń regularnych można wyszukiwać w tekście wystąpienia wzorca.

Wyrażenia regularne są reprezentowane w JavaScript za pomocą obiektów typu RegExp.

Slajd 20




Obiekt RegExp

Tworzenie obiektów RegExp

- `var zmienna = new RegExp(wzorzec, atrybuty)`
- `var zmienna = /wzorzec/atributy`

```
var wzorzec = /JavaScript/  
wzorzec.test("To jest kurs JavaScript");
```



Wyrażenia regularne są reprezentowane w JavaScript za pomocą obiektów typu RegExp.

Obiekt może zostać utworzony na dwa sposoby

- przez jawne wykorzystanie konstruktora obiektu (`var zmienna = new RegExp(wzorzec, atrybuty)`)
- przez przypisanie wyrażenia do zmiennej

Utwórzmy prosty wzorzec – niech to będzie ciąg znaków Javascript. Aby przetestować czy w tekście znajduje się ustalony przez nas wzorzec wywołujemy metodę `test` obiektu RegExp.

Metoda zwraca `true` jeśli wzorzec występuje w tekście i `false` w przeciwnym przypadku.

Obiekt RegExp ma jeszcze kilka innych metod:



Slajd 21



Metody obiektu RegExp

- `test(tekst)` `var wynik = wzorzec.test(tekst);`
- `compile(wyrażenie [, atrybuty])`
- `exec(tekst)` `var wynik = wzorzec.exec(tekst);`





Metoda `test` bada czy w zadanym jako argument tekście znajduje się wzorec, zdefiniowany przez obiekt typu `RegExp`. Wynikiem jest `true` lub `false`.

Metoda `compile()` dokonuje wewnętrznej kompilacji wyrażenia, co pozwala na szybsze jego przetwarzanie. Wyrażenie może zawierać argumenty i – pomijanie wielkości liter, g – globalne, m – przetwarzanie ma dotyczyć ciągu zawierającego wiele wierszy.

`exec` – metoda przeszukuje tekst w poszukiwaniu wzorca. Zwraca tablicę z wynikami dopasowań.

Slajd 22




Tworzenie wyrażeń

Atrybuty (flagi) wyrażeń regularnych

- g
- i
- m

```
var wzorzec = new RegExp("javascipt", "i");  
var wzorzec = /javascipt/i;
```



Wspomnieliśmy przed chwilą, że kompilowane wyrażenie może zawierać atrybuty, oraz że obie metody tworzenia wyrażeń także takie miejsce mają.

W wyrażeniach regularnych możemy podawać atrybuty (zwane też flagami) pozwalające na zmianę zachowania wyrażenia.



Dostępne są następujące parametry:

- g – globalny – oznaczający że przetwarzany ma być cały ciąg (bez tego znacznika) zwracany będzie tylko pierwszy dopasowany fragment. Parametr o bardzo dużym znaczeniu przy poszukiwaniu wszystkich wystąpień wzorca w tekście, można go pominąć jeśli interesuje nas tylko czy tekst zawiera fragment opisany wzorcem
- i – oznaczający że wielkość liter w tekście ma być ignorowana. Bez podania tego znacznika domyślną wartością jest `false` – treść w tekście musi mieć taką samą wielkość liter jak ta zdefiniowana we wzorcu
- m – oznaczający, że przetwarzanie ma dotyczyć tekstu wielowierszowego. W trybie tym znak początku i końca wzorca (^\$) jest wstawiany przed i po znaku

Moduł szkoleniowy JavaScript str. 91

nowej linii (\n).


Slajd 23

Tworzenie wyrażeń



Wyrażenia budowane są ze znaków zwykłych (litery alfabetu i cyfry) i znaków specjalnych (większość znaków interpunkcyjnych) takich jak:
`^ $. * + ? = ! : | \ / () [] { }`

Aby wykorzystywać znaki specjalne w wyrażeniach należy poprzedzić je znakiem lewego ukośnika (`\`)



Wyrażenia budowane są ze znaków zwykłych oraz znaków specjalnych. Znaki zwykłe to wszystkie litery alfabetu i cyfry. Znaki specjalne to większość znaków interpunkcyjnych takich jak: `^ $. * + ? = ! : | \ / () [] { }`. Aby z nich korzystać należy podać przed nimi znak `\` (lewy ukośnik)


Slajd 24

Tworzenie wyrażeń

W wyrażeniach regularnych wprowadzono pojęcie klasy znakowej.
 Klasa znakowa to zdefiniowany zestaw znaków, aby ją zbudować korzystamy z operatora zakresu `[]`.

var wyrażenie = `/[a-zA-Z]/`



W wyrażeniach regularnych wprowadzono pojęcie klasy znakowej. Klasa znakowa to zdefiniowany zestaw znaków, aby ją zbudować korzystamy z operatora zakresu `[]`. Istnieją klasy predefiniowane które zawierają definicje często wykorzystywanych klas



Slajd 25




Klasy znakowe

- [...] Dowolny znak znajdujący się w nawiasie
- [^...] Dowolny znak nieznajdujący się w nawiasie
- . Dowolny znak
- \d – dowolna cyfra
- \D – dowolny znak inny niż cyfra
- \s – dowolny biały znak (tabulator, spacja)
- \S – dowolny znak niebędący białym znakiem
- \w – dowolny znak wyrazu złożonego ze znaków ASCII
- \W – dowolny znak niebędący znakiem słowa złożonego ze znaków ASCII



[...] Dowolny znak znajdujący się w nawiasie – lista znaków podana w nawiasie jest listą znaków, które mogą wystąpić na tej pozycji w tekście

[^...] Dowolny znak nieznajdujący się w nawiasie – odwrotność – na pozycji w tekście nie mogą wystąpić znaki z nawiasu. Operator ^ musi być pierwszym po nawiasie

. Dowolny znak – na tej pozycji może być dowolny znak oprócz znaku nowej linii

\d – dowolna cyfra – uproszczony zapis [0-9]

\D – dowolny znak inny niż cyfra – odwrotność powyższego [^0-9]



\s – dowolny biały znak (tabulator, spacja)

\S – dowolny znak niebędący białym znakiem

\w – dowolny znak wyrazu złożonego ze znaków ASCII – dowolna cyfra, litera lub znak podkreślenia [0-9a-zA-Z_]

\W – dowolny znak niebędący znakiem słowa złożonego ze znaków ASCII [^0-9a-zA-Z_]

Slajd 26





Powtórzenie wzorca

Często zdarza się sytuacja, że fragment wzorca powinien się powtórzyć. Istnieją odpowiednie elementy, które umieszczone w wyrażeniu oznaczają ile razy może pojawić się dany element znajdujący się przed nim.

- ?
- *
- +

ab?c+d*



Często zdarza się sytuacja, że fragment wzorca powinien się powtórzyć. Istnieją odpowiednie elementy, które umieszczone w wyrażeniu oznaczają ile razy może pojawić się dany element znajdujący się przed nim.

? – oznacza że element może pojawić się 0 lub jeden raz



* - oznacza że element może pojawić się dowolną ilość razy (zero lub więcej)

+ - oznacza, że element może pojawić się jeden lub więcej razy

W powyższym wyrażeniu dopasowane zostaną fragmenty tekstu, które: zaczynają się od znaku "a", po nich może ale nie musi wystąpić litera "b", następnie będzie co najmniej jedna litera c po których może

wystąpić dowolna liczba liter d

Slajd 27


Powtórzenie wzorca

Dodatkowo dostępne są następujące operatory

- {n} – element ma się powtórzyć n razy
- {n,} – element ma się powtórzyć co najmniej n razy
- {n,m} – element ma się powtórzyć co najmniej n razy, ale nie więcej niż m razy

Przykład: kod pocztowy



```
var kod = /\d{2}-\d{3}/
/[0-9][0-9][0-9][0-9][0-9]/
```



Dodatkowo dostępne są następujące operatory

{n} – element ma się powtórzyć n razy
 {n,} – element ma się powtórzyć co najmniej n razy
 {n,m} – element ma się powtórzyć co najmniej n razy, ale nie więcej niż m razy

Slajd 28





Kwantyfikatory zachłanne i leniwe

- zachłanne – próbują dopasować się do jak najdłuższych podciągów znaków

```
var wyrażenie = /<.*>/
var wynik = wyrażenie.exec("aaa <br> bbb <br> ccc");
wyrażenie = /<[^>]*>/
```

- leniwe – powstają przez dodanie dodatkowego znaku zapytania {?, *, +, {n}?, {n,}?, {n,m}?



Wszystkie opisane wcześniej kwantyfikatory określone są jako zachłanne- będą próbowały dopasować się do jak najdłuższych podciągów.



Rozważmy taki przykład – wyrażenie ma wyszukiwać znaczniki html w tekście. Wiemy że znaczniki to fragmenty tekstu ujęte w nawiasy trójkątne.

Spodziewamy się wyników

 ale wyrażenie dopasowało jeszcze tekst
 bbb
. Fragment ten także pasuje do naszego wyrażenia, ale nie jest poprawnym znacznikiem html. Aby to poprawić należałoby użyć następującego wyrażenia: /<[^>]*>/ określi ono, że między nawiasami nie może znaleźć się nawias zamykający. Innym sposobem jest wykorzystanie

kwantyfikatorów leniwych, powstałych przez dodanie do zwykłych kwantyfikatorów dodatkowego pytajnika.


Slajd 29

Grupy

```

(abc)
(abc (123)?)
aaa (bbb|ccc|ddd)
:([^\s]+):1;
0testabc1testxyz
\d[a-z]{4}(?=abc)
\d[a-z]{4}(?!=abc)
  
```



Wcześniejsze metody pokazywały jak określić liczbę powtórzeń określonego znaku. Częstą sytuacją jest jednak powtarzanie się we wzorcu grup znaków. Aby oznaczyć grupę należy wpisać ją w nawiasy okrągłe. Pierwszy przykład pokazuje grupę składającą się z sekwencji abc. Drugi grupę składającą się z sekwencji abc, po której może wystąpić sekwencja cyfr 123. Cała grupa może pojawić się raz lub więcej.

Możemy także zdefiniować kilka różnych wersji ciągu. W przykładzie trzecim pokazano sekwencję aaa po której może wystąpić albo sekwencja bbb, ccc lub ddd. Takie warianty należy oddzielić od siebie znakiem pionowej linii.

Grupowanie daje nam dodatkowo możliwość odwołania się do ciągu odnalezionego w tekście. Każda grupa wyodrębniona za pomocą nawiasu ma swój kolejny numer. Możemy się do takiej grupy odwołać za pomocą wyrażenia \numer. W przykładzie sprawdzamy czy w ciągu wartości oddzielonych średnikiem nie ma dwóch takich samych umieszczonych obok siebie

Inną dodatkową możliwością jest spojrzenie w przód – sprawdzenie czy w dalszej części wyrażenia występuje lub nie występuje określony ciąg.

Stosujemy do tego operatory `?=` i `?!=`

W powyższym przykładzie w tekście

0testabc1testxyz w 1 przypadku zostanie zwrócony tekst 0test (cyfra, 4 znaki a-z a po nich występuje ciąg abc) a w drugim przypadku 1test – pierwszy fragment jest pominięty, ponieważ chcemy w wyniku otrzymać wyrażenie po którym nie następuje ciąg abc.

9.9.3 Ćwiczenia

Na slajdach zaprezentowano kilka podstawowych ćwiczeń wykorzystujących zaprezentowane możliwości języka. Pierwszym jest ćwiczenie zmieniające kolor przycisku (slajd 6) obrazujący wykorzystanie JS do zmiany wyglądu witryny www.

Drugim ćwiczeniem jest ćwiczenie wymagające napisania funkcji walidującej wartość wpisaną w pole formularza (slajd 13)

9.9.4 Opis założonych osiągnięć ucznia

Uczestnik po zakończeniu lekcji zna podstawowe sposoby wykorzystania języka JavaScript przy tworzeniu stron www. Potrafi za pomocą wyrażeń regularnych przeszukiwać treść strony.

9.10 Lekcja 10 – Biblioteka JQuery

9.10.1 Cel lekcji

Celem lekcji jest wprowadzenie do zaawansowanego, ale bardzo prostego w użyciu frameworka JavaScript – jQuery. Jest to biblioteka funkcji napisana w języku JavaScript rozszerzająca i upraszczająca część rozwiązań zaprezentowanych w poprzedniej lekcji.

9.10.2 Treść – slajdy z opisem

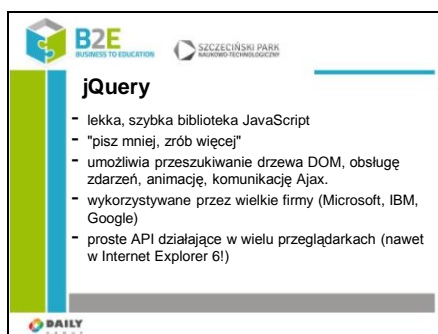
Slajd 1



W lekcji przedstawimy krótki opis frameworka jQuery, który jest jednym z najczęściej wykorzystywanym zbiorem funkcji w JavaScript.

Zaprezentujemy podstawowe mechanizmy jQuery, jego składnię a także kilka praktycznych przykładów wykorzystania w witrynach WWW.

Slajd 2



jQuery to lekka i szybka biblioteka JavaScript. "Write less, do more"



Zawiera następujące funkcjonalności:

- manipulację dokumentem html/ drzewem DOM
- manipulację CSS
- obsługę zdarzeń html
- efekty i animacje,
- AJAX,
- narzędzia
- system wtyczek pozwalający na rozszerzenie biblioteki

Zawiera proste do wykorzystania API, które działa w wielu przeglądarkach (obsługą różnic w interpretacji JavaScript zajęli się twórcy biblioteki)


jQuery jest wykorzystywane m.in. przez Wordpress (system do blogów, zarządzania treścią) oraz przez Wikipedię, Google, Microsoft itp

Slajd 3



Dodawanie jQuery

- Aby wykorzystywać jQuery musimy załączyć do naszej strony plik z kodem biblioteki.
- Możemy pobrać go ze strony www.jquery.com, lub wykorzystać mechanizm CDN – content delivery network.
- `<script src="jquery-1.9.1.min.js"></script>`
- `<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>`





Aby wykorzystywać jQuery musimy pobrać i załączyć do naszej strony plik z kodem biblioteki. Jest on dostępny na stronie www.jquery.com. Aktualna wersja to 1.9.1. Możemy pobrać plik w 2 wersjach zminimalizowanej lub normalnej. Wersja zminimalizowana ma usunięte wszystkie zbędne białe znaki. Jej celem jest maksymalne zmniejszenie rozmiaru pliku, co pozwala na szybsze załadowanie go do przeglądarki. Jeśli chcemy natomiast poznać budowę poszczególnych funkcji warto ściągnąć wersję normalną, bardziej czytelną dla człowieka.

Zamiast pobierać plik i podłączać go do naszej strony możemy wskazać odwołanie do skryptu wykorzystując mechanizm CDN – content delivery network. Są to specjalne serwisy przechowujące i udostępniające treści w internecie. Na slajdzie przedstawiłem przykład pobrania skryptu ze stron Google.

Zaletą podłączenia skryptu z CDN jest fakt, że w trakcie przeglądania internetu nasza przeglądarka już mogła pobrać jQuery (przy odwiedzeniu innej strony korzystającej z tej biblioteki) i sam plik został już umieszczony w pamięci podręcznej przeglądarki.

Slajd 4



Składnia jQuery

Podstawowa składnia to:


```
$(selector).action()
```

gdzie:

- znak \$ - wskazanie że wykorzystujemy jQuery,
- (selector) – wyrażenie do wyszukiwania elementu
- action() - akcja do wykonania

Przykłady:

- `$(this).hide()` – ukrywa bieżący element
- `$("p").hide()` – ukrywa wszystkie paragrafy
- `$(".test").hide()` – ukrywa wszystkie elementy o klasie test



W jQuery wyszukujemy element html i wykonujemy na nim akcje. Podstawowy zapis to `$(selector).action()`, gdzie znak \$ oznacza, że wykorzystujemy jQuery, (selector) – wyrażenie służące do wyszukiwania elementu, action() - akcja do wykonania



Przykłady:

`$(this).hide()` – ukrywa bieżący element

`$("p").hide()` – ukrywa wszystkie paragrafy

`$(".test").hide()` – ukrywa wszystkie elementy o klasie test

Slajd 5



zdarzenie Document Ready


Zdarzenie generowane po załadowaniu dokumentu.
Wykorzystywane aby opóźnić wywołanie funkcji jQuery
aż do momentu aż cała strona będzie dostępna

W skrypcie należy umieścić wywołanie:

```
$(document).ready(function() {  
    //tutaj nasze instrukcje  
});
```

lub wersję skróconą:

```
$(function() {  
    // tutaj instrukcje });
```



Zdarzenie generowane po załadowaniu dokumentu. Wykorzystywane aby opóźnić wywołanie funkcji jQuery aż do momentu aż cała strona będzie dostępna.

Przykładami niepoprawnych wywołań byłoby np. ukrycie elementu, który nie został jeszcze stworzony, pobranie rozmiaru obrazka, który nie został jeszcze pobrany itp.

W skrypcie należy umieścić wywołanie:

```
$(document).ready(function() {  
    //tutaj nasze instrukcje  
});
```

lub wersję skróconą:

```
$(function() {  
    // tutaj instrukcje });
```



Slajd 6

Selektory jQuery

- selektor elementu
`$("p")`
- selektor #id
`$("#test")`
- selektor .class
`$(".test");`
- selektor this
- selektor atrybutu np. [href]

Selektory jQuery są jedną z najważniejszych części biblioteki. Pozwalają na manipulację elementami html.

Wykorzystując selektory można znaleźć elementy html bazując na ich identyfikatorach, klasach, typach, atrybutach a nawet wartościach atrybutów.

Selektor elementu zwraca elementy o zadanej nazwie znacznika html, np. selektor `$("p")` zwróci wszystkie znaczniki p w dokumencie

Selektor identyfikatora zwraca elementy o danym identyfikatorze (atrybucie id) np. `$("#test")` zwróci element o id=test

Selektor klasy zwróci elementy o atrybucie class równym przekazanej wartości.

Selektor this zwraca bieżący element.

Selektor atrybutu [href] zwróci wszystkie elementy które zawierają atrybut href

Slajd 7

Zdarzenia

jQuery jest dostosowany do obsługi zdarzeń.
`$("p").click();`

Najczęściej wykorzystywane zdarzenia:

- `$(document).ready()`
- `click()`
- `mouseenter()`
- `mouseleave()`
- `hover`

jQuery jest dostosowany do obsługi zdarzeń. O zdarzeniach mówiliśmy na jednej z wcześniejszych lekcji. Jako przypomnienie zdarzenia pozwalają nam na reakcję na działania użytkowników na stronie. Tworząc odpowiednie metody obsługi zdarzeń możemy rozszerzać statyczne strony o interaktywne elementy.

Aby przypisać zdarzenie do elementu należy zastosować następującą składnię. Po selektorze elementu stawiamy . a po niej nazwę zdarzenia. Większość zdarzeń DOM ma swój odpowiednik w jQuery.

Najczęściej wykorzystywane zdarzenia:

`$(document).ready()` – odpalane po załadowaniu dokumentu

`click()` – odpalane po kliknięciu na element

`mouseenter()` – odpalane po najechaniu kursorem na element

`mouseleave()` – odpalane po

opuszczeniu elementu przez kursor
 hover() – ma 2 argumenty, jest
 połączeniem mouseenter i
 mouseleave

Slajd 8

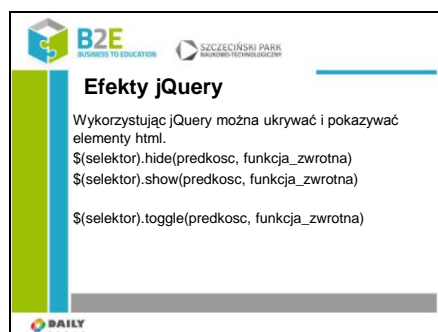


Efekty jQuery

Podstawowe efekty w jQuery:

- ukrywanie/pokazywanie
- zanikanie/rozświetlanie
- zwijanie/rozwijanie
- animacje

Slajd 9



Efekty jQuery

Wykorzystując jQuery można ukrywać i pokazywać elementy html.

```
$(selektor).hide(predkosc, funkcja_zwrotna)
$(selektor).show(predkosc, funkcja_zwrotna)
$(selektor).toggle(predkosc, funkcja_zwrotna)
```

Wykorzystując jQuery można ukrywać i pokazywać elementy html. Składnia poleceń jest następująca:

```
$(selektor).hide(predkosc, funkcja_zwrotna)
```

```
$(selektor).show(predkosc, funkcja_zwrotna)
```


prędkość to prędkość pokazywania/ukrywania elementu.

Możliwe są następujące wartości: fast, slow i czas podany w milisekundach.

opcjonalny parametr funkcja_zwrotna oznacza funkcję, która ma być uruchomiona po zakończeniu akcji.

Zamiast podłączać 2 akcje – jedną do ukrywania elementu, drugą do pokazywania możemy wykorzystać akcję toggle. Jeśli element był ukryty to zostanie pokazany, jeśli jest widoczny to zostanie ukryty

Slajd 10

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY**Efekty - zanikanie**

W jQuery można wykonać animację zanikania i pojawiania się elementu. Służą do tego akcje:



`fadeIn()`
`fadeOut()`
`fadeToggle()`
`fadeTo()`

 DAILY
GROUP

W jQuery można wywołać efekt zanikania i pojawiania się elementu. Wykorzystuje się do tego następujące metody:

`fadeIn()` – rozjaśnia ukryty element.
`fadeOut()` – element zanika z ekranu
`fadeToggle()` – przełącza między metodami `fadeIn` i `fadeOut`
`fadeTo()` – element pojawia się z określoną przezroczystością (wartość między 0 a 1)

Slajd 11

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY**Efekty – rozwijanie i zwijanie**

Efekt rozwijania i zwijania elementów można uzyskać za pomocą następujących akcji:

`slideUp()`
`slideDown()`
`slideToggle()`

 DAILY
GROUP



Efekt rozwijania i zwijania elementów można uzyskać za pomocą następujących akcji:

`slideUp()`
`slideDown()`
`slideToggle()`

Typy akcji są identyczne jak poprzednie.




Slajd 12



Animacje

Metoda `animate()` pozwala na utworzenie własnych animacji

```
$(selektor).animate(  
  {parametry},  
  predkosc,  
  funkcja_zwrotna);  
  
$("button").click(function() {  
  $("div").animate({left: 250px});  
});
```



Metoda `animate` pozwala na utworzenie własnych animacji. Wymagany argument "parametry" definiuje właściwości css, które będą animowane. Powyższy przykład po naciśnięciu przycisku przesunie element w lewo aż jego właściwość `left` osiągnie 250 pikseli.

Domyślnie elementy HTML mają właściwość `static` – nie można ich przesuwac. Przed zdefiniowaniem animacji trzeba ustawić właściwość `position` elementu na `relative`, `fixed` lub `absolute`.



Można w parametrach przekazać więcej niż 1 właściwość. Możemy stworzyć animację, która przesunie element i zwiększy jego rozmiar. Jak powinna wyglądać taka funkcja?

Za pomocą metody `animate` można zmieniać większość właściwości stylu css – trzeba tylko pamiętać o zmianie trybu pisania na camel case. zamiast pisać `padding-left` napiszemy `paddingLeft` itd. Podstawowa biblioteka jQuery nie obsługuje animacji atrybutu `color`. Aby animować zmiany kolorów trzeba pobrać odpowiednią wtyczkę z witryny jquery. jQuery automatycznie tworzy kolejkę dla animacji. Możemy zdefiniować kilka animacji a jQuery automatycznie wyświetli je jedna po drugiej.

Animacje można zatrzymać wykorzystując metodę `stop` z opcjonalnymi argumentami `stopAll` (powoduje zatrzymanie całej kolejki animacji) i `goToEnd` – przechodzi na koniec animacji.


Oba argumenty mają domyślną wartość `false`, co oznacza że zatrzymana zostanie tylko bieżąca animacja.

Slajd 13





Funkcje zwrotne

Wyrażenia JavaScript są przetwarzane jedno po drugim.
Jednak w przypadku efektów, kolejna linia może zostać przetworzona zanim efekt się zakończy.
Funkcje zwrotne to funkcje, które zostaną wykonane, gdy działanie efektu się zakończy.



Wyrażenia JavaScript są przetwarzane jedno po drugim. Jednak w przypadku efektów, kolejna linia może zostać przetworzona zanim efekt się zakończy. Funkcje zwrotne to funkcje, które zostaną wykonane, gdy działanie efektu się zakończy.


Slajd 14



Łączenie akcji


Łączenie akcji pozwala na wywołanie wielu metod jQuery na danym elemencie za pomocą jednego wyrażenia. W ten sposób przeglądarka nie musi wyszukiwać danego elementu po raz kolejny. Aby połączyć akcje należy ją po prostu dodać po wywołaniu poprzedniej akcji.

```
$("#p1").css("color", "red")  
    .slideUp(2000).slideDown(2000);
```



Łączenie akcji pozwala na wywołanie wielu metod jQuery na danym elemencie za pomocą jednego wyrażenia. W ten sposób przeglądarka nie musi wyszukiwać danego elementu po raz kolejny. Aby połączyć akcje należy ją po prostu dodać po wywołaniu poprzedniej akcji. Przykład obrazuje działanie na obiekcie o id #p1 – zmieniamy jego kolor na czerwony, po czym zwiijamy obiekt a następnie rozwijamy. Operacje zwiijania i rozwijania trwają po 2 sekundy

Slajd 15



Manipulacja DOM

jQuery zawiera metody do manipulacji elementami html i ich atrybutami. Wykorzystuje do tego metody działające na modelu dokumentu (DOM)
DOM – interfejs, który pozwala programom i skryptom na dynamiczny dostęp i modyfikację treści, struktury i stylu dokumentu



jQuery zawiera metody do manipulacji elementami html i ich atrybutami. Wykorzystuje do tego metody działające na modelu dokumentu (DOM)
Jak wspominaliśmy wcześniej DOM to interfejs, który pozwala programom i skryptom na dynamiczny dostęp i modyfikację treści, struktury i stylu dokumentu

Slajd 16

Pobieranie treści text() html()

- text()

```
$("#btn1").click(function(){
  alert("Text: " + $("#test").text());
});
```
- html()

```
$("#btn2").click(function(){
  alert("HTML: " + $("#test").html());
});
```

Podstawowymi metodami do pobierania treści dokumentu są metody:

text() – pobiera lub ustawia wartość tekstową wybranego elementu.

html() – pobiera lub ustawia treść elementu (razem ze znacznikami html)

Slajd 17

Metody val() i attr()

- val()

```
$("#btn1").click(function(){
  alert("Value: " + $("#test").val());
});
```
- attr()

```
$("#button").click(function(){
  alert($("#link").attr("href"));
});
```

val() – pobiera lub ustawia wartość pola formularza.

Metoda attr() pobiera wartość danego atrybutu

Wszystkich wymienionych teraz funkcji można do ustawienia wartości danego elementu. Wartość tę należy umieścić w nawiasie jako parametr metody. Zamiast wartości możemy podać funkcję zwrótną, która jako argumenty przyjmuje indeks bieżącego elementu w zaznaczeniu, oraz wartość oryginalną elementu. Jako wartość wynikową ustawiamy tekst, który ma być nową wartością elementu

Slajd 18

Dodawanie elementów

- append()

```
$(p).append("dodano z jQuery.");
```
- prepend()

```
$(p).prepend("dodano z jQuery.");
```
- after()
- before()

Za pomocą jQuery można bardzo łatwo dodawać elementy i treść do strony. Służą do tego następujące metody:



- append() – dodaje treść na końcu danego elementu.

Załóżmy, że mamy document zawierający jakiś paragraf o treści Witaj świecie!. Wywołanie metody \$(p).append("dodano z jQuery."); spowoduje, że paragraf ten będzie miał teraz treść: Witaj świecie! dodano z jQuery.;

- prepend() – dodaje treść na początku danego elementu. W powyższym przykładzie wynikiem będzie tekst: "dodano z jQuery.Witaj świecie!"

Metody `after` i `before` wstawiają treść odpowiednio przed i po elemencie – nie tak jak wcześniejsze na początku lub końcu wartości elementu

Slajd 19

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY

Usuwanie elementów

Do usuwania elementów służą metody:

- `remove()`
`$("#div").remove()`
`$("#p").remove(".italic");`
- `empty()`
`$("#div").empty();`



Do usuwania elementów służą metody:

- `remove()`
- `empty()`

Metoda `remove` usuwa wszystkie węzły potomne elementu oraz sam element, jako parametr możemy podać selektor elementów do usunięcia. W przykładzie usuwamy wszystkie elementy `p`, które mają klasę `italic`.

Metoda `empty` usuwa wszystkie węzły potomne danego elementu.

Slajd 20

 **B2E**
BUSINESS TO EDUCATION  SZCZECIŃSKI PARK
NAUKOWO-TECHNOLOGICZNY

Manipulacja CSS

Do zmiany stylu css elementów służą następujące metody:



- `addClass()`
- `removeClass()`
- `toggleClass()`
- `css()`

Do zmiany stylu css elementów służą następujące metody:

- `addClass()` – metoda pozwala na dodanie klasy do wybranego elementu na stronie. Jako parametr podajemy nazwę klasy css zdefiniowaną w pliku stylu. Możemy dodać od razu kilka klas
- `removeClass()` – metoda usuwa podaną klasę z elementu.
- `toggleClass()` – metoda dodaje lub usuwa klasę z elementu
- `css()` – ustawia lub zwraca jeden lub wiele właściwości stylu dla wybranego elementu;
aby pobrać wartość elementu należy wywołać metodę z nazwą właściwości, aby ustawić – oprócz nazwy podajemy nową wartość. W przypadku


pobierania wartości zostanie zwrócona wartość pierwszego dopasowanego obiektu do selektora. Ustawienie wartości spowoduje zmiany we wszystkich elementach wyszukanych przez selektor.

Slajd 21

jQuery i AJAX

- co to jest AJAX?
Asynchronous JavaScript and XML
- w skrócie: służy do pobierania danych w tle i wyświetlania ich na stronie bez konieczności przeładowania strony




-co to jest AJAX? Asynchronous JavaScript and XML.

- w skrócie: służy do pobierania danych w tle i wyświetlania ich na stronie bez konieczności przeładowania strony.

Pisanie wywołań AJAX jest utrudnione, różne przeglądarki inaczej implementują AJAX. Oznacza to konieczność napisania dodatkowego kodu dla różnych przeglądarek.

W jQuery funkcjonalność ta już została zaimplementowana i można jej używać używając pojedynczej linii kodu


Slajd 22




metoda load()

Metoda load() pobiera dane z serwera i umieszcza je w wybranym elemencie

```
$(selektor).load(url, dane, funkcja_zwrotna)
$("#div1").load("dane.txt");
```



Metoda load() pobiera dane z serwera i umieszcza je w wybranym elemencie.

Polecenie może mieć 3 parametry:

- obowiązkowy URL, który wskazuje adres serwera (pliku lub usługi),
- opcjonalne dane: zawierający informacje które chcemy wysłać razem z żądaniem (np. identyfikatory, nagłówki autoryzacji itp.)
- opcjonalne funkcję_zwrotną, która zostanie wywołana po odebraniu danych.

Funkcja zwrotna ma 3 parametry: responseTxt – tekst odpowiedzi, statusTXT – zawiera status żądania,

xhr – zawiera obiekt XMLHttpRequest
Założmy że mamy na dysku plik dane.txt zawierający dane, które chcemy wyświetlić w elemencie div1. Użyjemy do tego polecenia `$("#div1").load("dane.txt");`

Slajd 23



jQuery get() i post()

Metody get() i post()

```
$.get(URL, funkcja_zwrotna);  
$.post(URL, dane, funkcja_zwrotna);
```

Metody get i post wysyłają żądania do serwera za pomocą żądań HTTP GET i HTTP POST.

GET – pobiera dane z określonego adresu

POST – wysyła dane na określony adres

Funkcja get() przyjmuje 2 parametry:

- obowiązkowy URL – adres zasobu, który chcemy pobrać
- i opcjonalny funkcja_zwrotna (callback) – funkcja wywołana po odebraniu danych

Funkcja post() przyjmuje 3 argumenty:

- obowiązkowy URL – adres na który wysyłamy żądanie.

- opcjonalny dane - dane które chcemy wysłać
- opcjonalny funkcja_zwrotna – funkcja, która zostanie uruchomiona po wysłaniu danych



Slajd 24

Podsumowanie

W trakcie tej lekcji poznaliśmy podstawowe informacje o frameworku jQuery. Wiemy, że pozwala on m.in. na:

- animowanie elementów na stronie (pokazywanie, ukrywanie, przesuwanie itp.)
- wyszukiwanie i modyfikację treści elementów
- dodawanie i usuwanie elementów
- modyfikację stylu css strony
- komunikację z innymi usługami z wykorzystaniem AJAX

9.10.3 Ćwiczenia

Napisz skrypt dodający efekt "zebry" w tabeli (wiersze parzyste i nieparzyste mają inne kolory). Dodatkowo niech wiersz nagłówkowy zawiera test pogrubiony.

Utwórz stronę zawierającą prosty formularz (zawierający jedno pole o identyfikatorze "name") i element div. Po wprowadzeniu znaku do pola dodaj wpisany znak do utworzonej warstwy. Wykorzystaj zdarzenie keyup.

Fragment przykładowego rozwiązania (część jQuery):

```
<script type="text/javascript">

$(document).ready(function() {

    $("#name").bind('keyup', function() {

        $("#div").append('').append($(this).val());

    });

});

</script>
```

9.10.4 Opis założonych osiągnięć ucznia

Uczestnik po zakończeniu lekcji potrafi dodać do strony bibliotekę jQuery, a także wykorzystać ją podczas tworzenia witryny www. Potrafi dynamicznie zmieniać wygląd strony a także dodać podstawowe efekty na stronie.