informatyka+

Algorytmika i programowanie

Bazy danych

Multimedia, grafika i technologie internetowe

Sieci komputerowe

Tendencje w rozwoju informatyki i jej zastosowań

Człowiek – najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

informatyka+

Wszechnica Poranna: Tendencje w rozwoju informatyki i jej zastosowań

Do czego można wykorzystać język JavaScript

Krzysztof Ciebiera

Człowiek – najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Do czego można wykorzystać język JavaScript

Rodzaj zajęć: Wszechnica Poranna
Tytuł: Do czego można wykorzystać język JavaScript
Autor: mgr Krzysztof Ciebiera

Redaktor merytoryczny: prof. dr hab. Maciej M Sysło

Zeszyt dydaktyczny opracowany w ramach projektu edukacyjnego **Informatyka+** — ponadregionalny program rozwijania kompetencji uczniów szkół ponadgimnazjalnych w zakresie technologii informacyjno-komunikacyjnych (ICT).
www.informatykaplus.edu.pl
kontakt@informatykaplus.edu.pl

Wydawca: Warszawska Wyższa Szkoła Informatyki
ul. Lewartowskiego 17, 00-169 Warszawa
www.wwsi.edu.pl
rektorat@wwsi.edu.pl

Projekt graficzny: FRYCZ I WICHA

Warszawa 2009

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Do czego można wykorzystać język JavaScript

Krzysztof Ciebiera
Wydział Matematyki, Informatyki i Mechaniki,
Uniwersytetu Warszawskiego
ciebie@mimuw.edu.pl

Streszczenie

Zajęcia mają na celu przedstawienie możliwości języka JavaScript (JS) do wzbogacania stron i serwisów WWW o elementy interaktywne i graficzne. Na wykładzie zostaną zademonstrowane efekty wykorzystania fragmentów języka JS na stronach internetowych. Druga część wykładu będzie łagodnym wprowadzeniem do podstawowych konstrukcji tego języka, omówione zostaną: zmienne, instrukcje iteracyjne i warunkowe, liczby, napisy i tablice.

Warsztaty będą okazją do praktycznego przećwiczenia wprowadzonych na wykładzie elementów języka JS. Głównym celem zajęć praktycznych będzie utworzenie własnego programu do zabawy w Sudoku. Program będzie mógł być zintegrowany z serwisem, na którym np. mogłyby być organizowane konkursy Sudoku.

Do wzięcia udziału w tych zajęciach przydatna będzie, ale nie jest wymagana, elementarna znajomość języka HTML i programowania w jakimkolwiek języku.

Planuje się zorganizowanie dla uczestników Projektu Informatyka + konkursu serwisów internetowych do rozgrywek w Sudoku.

Spis treści

1 WSTĘP

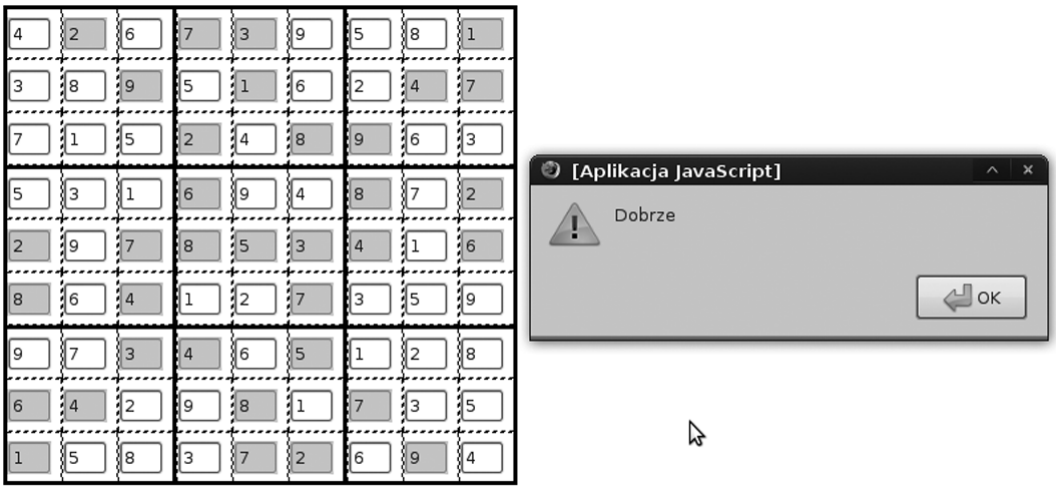
W trakcie warsztatów przygotujemy przy użyciu języka JavaScript program do zabawy w Sudoku. Program zrealizujemy w taki sposób, aby możliwe było w miarę łatwe jego zintegrowanie z większym serwisem, na którym np. mogłyby być organizowane konkursy Sudoku.

Co to jest Sudoku

Sudoku jest łamigłówką logiczną. Na kwadratowej planszy 9x9, w niektórych jej polach są wpisane liczby od 1 do 9. Należy uzupełnić planszę liczbami od 1 do 9 tak, aby w żadnym wierszu, w żadnej kolumnie, ani w żadnym z dziewięciu głównych kwadratów (patrz rys. 1) nie powtarzała się ta sama liczba i aby wszystkie kwadraty były wypełnione.

Jaki jest nasz cel

Postaramy się zaprogramować planszę pokazaną na rys. 1. :



Rysunek 1.
Przykładowa, wypełniona poprawnie plansza Sudoku

Przyjmujemy, że nasz program będzie miał następujące funkcjonalności:

1. Działa w przeglądarce WWW.
2. Wyświetla zagadkę Sudoku.
3. Umożliwia wprowadzanie rozwiązania.
4. Sprawdza, czy rozwiązanie jest poprawne.

2 WYMAGANIA W STOSUNKU DO UCZESTNIKÓW I SPRZĘTU

CO NAM BĘDZIE POTRZEBNE DO NAPISANIA PROGRAMU

Wyposażenie i wymagania niezbędne do realizacji postawionego zadania:

1. Przeglądarka WWW, najlepiej Firefox, ze względu na to, że w notatkach do warsztatów używamy programu Firebug, ale może być dowolna inna przeglądarka, byleby tylko uczestnik warsztatów umiał w niej debugować.
2. Dostęp do Internetu w celu zainstalowania oprogramowania jQuery i Firebug.
3. Edytor tekstu. Najlepszy byłby edytor podkreślający składnię HTML i JavaScript (np. Geany), ale może być też notatnik w Windows.

Co powinniśmy wcześniej umieć

Niezbędne są dwie rzeczy:

1. Znajomość HTML w podstawowym zakresie, czyli: co to jest tabelka, formularz, odnośnik i jak to wszystko zapisać w postaci strony HTML.
2. Znajomość podstaw programowania w dowolnym języku w zakresie: co to jest zmienna, pętla i funkcja. Składnię i tak będziemy poznawać na bieżąco w miarę potrzeb.

3 PRZEBIEG WARSZTATÓW

3.1 UŻYWANIE JĘZYKA JS

Nauczymy się podstaw korzystania z języka JavaScript(JS). Spróbujmy policzyć, ile to jest 3*(2+1). W tym celu:

- otwieramy edytor tekstu w którym można edytować źródła HTML (np. Notatnik, edytor vim lub podobny); wklejamy do niego tekst, który jest poniżej; .
- zapisujemy ten tekst w pliku pod nazwą kalkulator.html;
- otwieramy ten plik w przeglądarce.

Oto tekst źródłowy kalkulatora:

```
<html>
 <head><title>kalkulator</title></head>
 <body>
  <script type="text/javascript">
   alert(3*(2+1));
  </script>
 </body>
</html>
```

Jeśli wszystko zrobiliśmy dobrze, to na stronie powinno się pokazać okienko z napisem 9, bo 3*(2+1) = 9. Zauważmy, że to proste ćwiczenie przekonuje nas, iż potrafimy już:

1. Napisać program w JS.
2. Wstawić program w JS do dokumetu w języku HTML.
3. Uruchomić program w JS.
4. Wyświetlić źródłowy tekst programu (klawisze Ctrl+U w przeglądarce Firefox).
5. Uruchomić program jeszcze raz – w tym celu przeładowujemy stronę (klawisze Ctrl+R w Firefox).

Zadania do samodzielnego wykonania

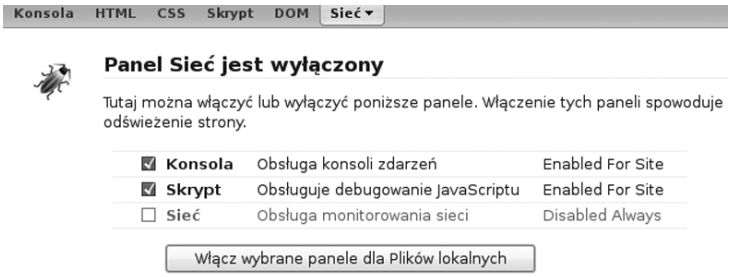
1. Sprawdź, czy poprawnie są wykonywane pozostałe operacje arytmetyczne (+, i, *, /).
2. Ponadto, sprawdź:

- Ile to jest (11 % 3) i dlaczego?
- Ile to jest (11 / 3) i dlaczego?
- Ile to jest (Math.round(11/3))?
- Ile to jest (3+1==4)?
- Ile to jest (2+4/7>1)?

3.2 INSTALUJEMY PROGRAM FIREBUG

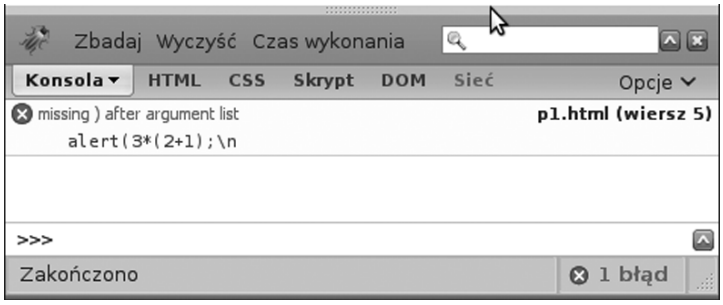
Mogłoby się wydawać, że teraz powinniśmy zacząć pisać nasze Sudoku, ale wstrzymajmy się z tym na chwilę. Zainstalujmy najpierw program Firebug, bedący rozszerzeniem przeglądarki Firefox, który pomoże nam w **debugowaniu** naszego programu, czyli w znajdowaniu i poprawianiu w nim błędów. Aby zainstalować program Firebug, należy wejść na stronę: https://addons.mozilla.org/pl/firefox/addon/1843, nacisnąć przycisk **Instaluj** i postępować zgodnie z instrukcją.

Po zrestartowaniu przeglądarki zauważymy małego robaczka w prawym dolnym rogu – to oznacza, że program Firebug został zainstalowany. Zmieńmy kod programu i wpiszmy `alert(3*(2+1);` – usunęliśmy ostatni nawias zamykajacy. I nic się nie stało. To dlatego, że Firebug nie wie, że chcemy, żeby działał. Klikamy na jego ikonce (ten robaczek) i zaznaczamy pola **Konsola** i **Skrypt**, a następnie klikamy przycisk **Włącz wybrane panele dla Plików lokalnych** – patrz rys. 2.



Rysunek 2.
Uaktywnienie programu Firebug

Po tej zmianie zauważymy, że w naszym skrypcie jest jeden błąd, a po jego kliknięciu na polu, w którym jest wyświetlana informacja o błędzie zobaczymy, na czym ten błąd polega i w którym jest wierszu, patrz rys. 3.



Rysunek 3.
Sygnalizacja błędu w skrypcie JS

Zakładam, że od tej pory uczestnik warsztatów sam będzie sobie radził z błędami w składni JS.

3.3 INSTALUJEMY BIBLIOTEKĘ JQUERY

Nie będziemy pisać w „czystym" języku JavaScript, tylko skorzystamy z pomocy biblioteki jQuery. Ściągamy ją ze strony http://jquery.com/ klikając przycisk Download(jQuery), a potem jeszcze raz na nazwie pliku z aktualną wersją biblioteki (np.jquery-1.3.2.min.js). Następnie zmieniamy nazwę ściągniętego pliku na jquery.js i umieszczamy go w katalogu z innymi plikami, nad którymi pracujemy. Nasze Sudoku będzie korzystało z jQuery.

Aby sprawdzić, czy wszystko jest w porządku, zaczniemy tworzyć nasze Sudoku. Niech ma ono na razie następującą postać:

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <title>Sudoku</title>
  <script type="text/javascript" src="jquery.js"></script>
   <script type="text/javascript">
   $(document).ready(function(){
     alert("Wczytane");
   })
   </script>
 </head>
 <body>
  <div id='plansza'>Miejsce na planszę</div>
 </body>
</html>
```

Jeśli wszystko zrobiliśmy poprawnie, to po wczytaniu całej strony zostanie wyświetlone okienko z napisem `Wczytane`. Jeśli coś jest źle, to sprawdź za pomocą programu Firebug, czy nie popełniłeś błędu. Może nie przegrałeś w dobre miejsce pliku jquery.js? Może źle się on nazywa?

W jaki sposób jest wyświetlany napis `Wczytane`?

```
$(document).ready(function(){
    alert("Wczytane");
})
```

JQuery udostępnia funkcję `$(document).ready(function(){ ... }`, gdzie w miejsce trzech kropek możemy wpisać dowolny program w JS, który powinien zostać uruchomiony po załadowaniu strony. W tym przypadku ten program składa się z jednej instrukcji `alert("Wczytane")`, która powoduje wyświetlenie okienka z napisem `Wczytane`. Teraz powinniśmy się zabrać za wyświetlenie planszy.

3.4 ZASTĘPOWANIE ZAWARTOŚCI ELEMENTÓW

Najprościej byłoby przygotować planszę w języku HTML. Należałoby zrobić tabelę, składającą się z dziewięciu wierszy, każdy z nich składający się z dziewięciu kolumn a w każdej z kolumn jedno pole do wprowadzania wyników. Czyli należałoby napisać coś takiego:

```
<table border=1>
 <tr><td><input ...><td>.....</tr>
  .....
</table>
```

Widać, że utworzenie takiej tabeli jest bardzo proste. Załóżmy, że nasza plansza byłaby mniejsza, np. taka:

```
<table border=1><tr><td>1<td>2<tr><td>3<td>4</table>
```

Gdybyśmy od kogoś dostali zmienną, zawierającą taką planszę w postaci napisu, to naszym jedynym zadaniem byłoby spowodowanie, aby została wyświetlona plansza w bloku `<div id='plansza'>`. Jak to zrobić w JS? Zamieńmy skrypt, który mamy na następujący:

```
<script type="text/javascript">
 plansza = "<table border=1><tr><td>1<td>2<tr><td>3<td>4</table>"
 $(document).ready(function(){
    $("#plansza").html(plansza);
 })
</script>
```

i przeładujmy stronę. W tej chwili już nie pokazuje się okienko z napisem `Wczytane`. Dzieje się coś dużo ciekawszego – jest wyświetlana nasza plansza. Jak do tego doszło?

W wierszu skryptu na zmienną `plansza` jest przypisywana wartość `<table> ....</table>`. W wierszach 2.-4. jest tworzona funkcja, która zostanie uruchomiona po wyświetleniu HTML. W wierszu 3. zawartość elementu o identyfikatorze `plansza`, jest zastępowana przez wartość zmiennej `plansza`.

Konstrukcja `$("#identyfikator").html(wartość)`, ustawia `wartość` elementu o identyfikatorze (atrybut `id` w HTML) `identyfikator`.

Teraz tylko potrzebujemy zbudować planszę.

3.5 PĘTLA FOR

Wyobraźmy sobie, jak nasza plansza mogłaby wyglądać? Dobrze, gdyby była kwadratowa i gdyby każde jej pole można było jednoznacznie zidentyfikować. Powiedzmy, że pierwszy wiersz zawiera pola z identyfikato-

rami 11, 12, 13, ..., 19. Drugi 21, 22, ..., 29, i tak dalej, aż do 91, 92, ..., 99. Aby zbudować taką planszę potrzebujemy dziewięć wierszy po dziewięć kolumn każdy. Aby powtórzyć jakąś operację dziewięć razy wykorzystamy pętlę `for`, która w JS ma taką samą postać jak w języku Java czy C:

```
for(i = 1; i <= 9; i++)
```

Ta pętla zostanie wykonana dziewięć razy z wartościami `i` = 1, 2, 3, ..., 9. Popatrzmy na program:

```
$(document).ready(function(){
 plansza = "<table border=1>"
 for(i = 1; i <= 9; i++)
 {
  plansza += "<tr>";
  for(j = 1; j <= 9; j++)
  {
   plansza += "<td>";
   plansza += i * 10 + j;
   plansza += "</td>";
  }
  plansza += "</tr>";
 }
 plansza += "</table>"
 $("#plansza").html(plansza);
})
```

Ten program zawiera dwie pętle `for`. Zewnętrzną, sterowaną zmienną `i` oraz wewnętrzną – sterowaną zmienną `j`. Zewętrzna pętla buduje wiersze, a wewnętrzna – kolumny. Używamy w tym programie kilku ciekawych konstrukcji:

1. `plansza += <napis>` – ta instrukcja dokleja `napis` do planszy.
2. `plansza += <liczba>` – ponieważ plansza jest napisem, to `liczba` zostanie również skonwertowana do napisu i zostanie doklejona do planszy.
3. `i++` – ta konstrukcja językowa powoduje zwiększenie wartości `i` o jeden. Oznacza ona to samo, co `i = i + 1`.
4. `for(i = 1; i <= 9; i++)` – to jest pętla `for`. Najpierw wartość `i` jest ustawiana na jeden. Następnie sprawdzane jest, czy `i` jest mniejsze lub równe dziewięć. Jeśli tak, to jest wykonywana instrukcja pętli i wartość`i` jest zwiększana o jeden. I znowu jest sprawdzane, czy `i <= 9`, jeśli tak, to wykonywana jest pętla, `i` jest podnoszone o jeden itd.

Zadania do samodzielnego wykonania

1. Jaką wartość ma zmienna `plansza` przed przypisaniem jej do elementu `#plansza`? Możesz to sprawdzić na co najmniej dwa sposoby, używając instrukcji `alert` lub używając debuggera wbudowanego w program Firebug.
2. Dodaj do planszy kolumnę i wiersz z numeracją.
3. Wyróżnij (np. za pomocą szarego tła) co drugi wiersz.

3.6 INSTRUKCJA IF

Gdy chcemy sprawić, aby fragment programu wykonał się jedynie po spełnieniu jakiegoś warunku, używamy instrukcji `if`. Jej składnia jest podobna do składni w językach C i Java:

```
if (a == 0)
{
 alert('a jest równe zero!);
}
```

3.7 FUNKCJE

Jeśli będziemy nasz program rozwijać w podobny sposób, to za chwilę stanie się nieczytelny. Dlatego podzielimy go na logiczne fragmenty przy użyciu **funkcji**. Funkcje w JS zachowują się inaczej niż w typowych językach programowania. Nie jest sprawdzane, czy do funkcji jest przekazywana odpowiednia liczba parametrów. Funkcja może zwracać swój wynik, ale nie musi. Nie jest sprawdzany typ tego wyniku, czyli funkcja może czasem zwracać liczbę, a czasem napis.

Wydzielmy teraz z naszego programu funkcję budującą planszę:

```
function Plansza(rozmiar)
{
 plansza = "<table border=1>"
 for(i = 1; i <= rozmiar; i++)
 {
  plansza += "<tr>";
  for(j = 1; j <= rozmiar; j++)
  {
   plansza += "<td>";
   plansza += i * 10 + j;
   plansza += "</td>";
  }
  plansza += "</tr>";
 }
 plansza += "</table>"
 return plansza;
}
$(document).ready(function(){
    $("#plansza").html(Plansza(6));
 })
```

W naszym programie pojawiła się funkcja `Plansza`. Pobiera ona jeden parametr `rozmiar` i zwraca zbudowaną planszę za pomocą instrukcji `return`.

3.8. ZMIENNE I ICH ZASIĘG

Moglibyśmy teraz wpaść na pomysł, aby podzielić funkcję budującą planszę na dwie części, z których jedna będzie budować wiersze, a druga składać je w całość. Przyjrzyjmy się następującemu programowi:

```
function Wiersz(numer,rozmiar)
{
 wiersz = "<tr>";
 for(i = 1; i <= rozmiar; i++) //UWAGA: zmieniliśmy j na i!
 {
  wiersz += "<td>";
  wiersz += numer * 10 + i;
  wiersz += "</td>";
 }
 wiersz += "</tr>";
 return wiersz;
}
function Plansza(rozmiar)
{
 plansza = "<table border=1>"
 for(i = 1; i <= rozmiar; i++)
 {
  plansza += Wiersz(i, rozmiar)
```

```
 }
 plansza += "</table>"
 return plansza;
}
$(document).ready(function(){
    $("#plansza").html(Plansza(6));
})
```

Ten program wygląda całkiem poprawnie, prawda? Niestety ma jedną wadę. Źle działa. Co się dzieje? Otóż wszystkie zmienne, których używamy wewnątrz funkcji są globalne. To znaczy, że zmienna `i` używana w pętli `for` w funkcji `Plansza`, to ta sama zmienna `i`, która jest używana w pętli `for` w funkcji `Wiersz`. W funkcji `Wiersz` ta zmienna zostanie podniesiona do wartości `6` i pętla w funkcji `Plansza` zostanie wykonana tylko raz. Podobnie zmienne `wiersz` i `plansza` są globalne! Co z tym zrobić? W języku JS do zdefiniowania zmiennej lokalnej służy słowo kluczowe `var`. Poprawmy zatem nasz program:

```
<script type="text/javascript">
function Wiersz(numer,rozmiar) //aa
{
 var wiersz = "<tr>";
 for(var i = 1; i <= rozmiar; i++)
 {
  wiersz += "<td>";
  wiersz += numer * 10 + i;
  wiersz += "</td>";
 }
 wiersz += "</tr>";
 return wiersz;
}
function Plansza(rozmiar)
{
 var plansza = "<table border=1>"
 for(var i = 1; i <= rozmiar; i++)
 {
  plansza += Wiersz(i, rozmiar)
 }
 plansza += "</table>"
 return plansza;
}
$(document).ready(function(){
    $("#plansza").html(Plansza(6));
})
</script>
```

Teraz ten program powinien już działać poprawnie. Zwróćmy uwagę na konstrukcję `$("#plansza").html(Plansza(6))`; Wywołuje ona funkcję `Plansza` z parametrem `6`, a jej wynik przekazuje do funkcji `html`, która ustawia go jako zawartość elementu `#plansza`.

Zadania do samodzielnego wykonania

1. Zmodyfikuj nasz program tak, aby wyświetlał tabliczkę mnożenia (z nagłówkiem).
2. Napisz program, który wyświetli trzy plansze o bokach 5, 7 i 9, jedna pod drugą.

3.9 OBSŁUGA NAPISÓW I LICZB

Czas zastanowić się, w jaki sposób będziemy przechowywać nasze Sudoku. Ze względów technicznych (łatwość przesyłania po sieci) dobrym formatem jest przechowywanie Sudoku w postaci 81 znakowego napisu (czytanego

od lewej do prawej z góry w dół) z zerami w miejscach, które pozostają do wypełnienia przez użytkownika. Opis Sudoku z rys. 1 wraz z funkcją budującą planszę mógłby mieć następująca postać:

```
var opisPlanszy =
"020730001" +
"009010047" +
"000208900" +
"000600802" +
"207853406" +
"804007000" +
"003405000" +
"640080700" +
"100072090";

function wspolrzedne(i)
{
 if (i < 0 || i > 80) alert("wspolrzedne: zle dane");
 return Math.floor(i/9) + 10 * (i % 9) + 11;
}

function Plansza(opis)
{
 var i;
 plansza = "<table>";
 for(i = 0; i < opis.length; i++)
 {
  if (i % 9 == 0) plansza += "<tr>";
  plansza += "<td class='field' id='td"+ wspolrzedne(i) +"'>";
  if (opis[i] == '0')
  {
   plansza += "<input type='text' size='1' maxlength='1' id='i"
    + wspolrzedne(i) + "'>";
  }
  else
  {
   plansza += "<input type='text' size='1' maxlength='1' readonly= '' id='i"
    + wspolrzedne(i) + "' value='"+opis[i]+"'>";
  }
  plansza += "</td>";
  if (i % 9 == 8) plansza += "</tr>";
 }
 return plansza+"</table>";
};

$(document).ready(function(){
    $("#plansza").html(Plansza(opisPlanszy));
})
```

Math

`Math` (formalnie jest to obiekt) udostępnia w JS następujące funkcje:

- `Math.abs(a)` // wartość bezwzględna a
- `Math.ceil(a)` // sufit a
- `Math.floor(a)` // podłoga a
- `Math.max(a,b)` // maksimum z a i b

- `Math.min(a,b)` // minimum z a i b
- `Math.pow(a,b)` // a do potęgi b
- `Math.random()` // pseudolosowa liczba od 0 to 1
- `Math.round(a)` // zaokrąglenie a
- `Math.sqrt(a)` // pierwiastek kwadratowy z a

W funkcji `wspolrzedne` korzystamy z `Math.floor()`, aby policzyć współrzędne i-tej komórki.

Napisy

JS udostępnia m.in. następujące funkcje do obsługi napisów:

```
napis = "ab" + "ca"     // łączenie (konkatenacja) napisów
napis == 'abca'         // porównanie true
napis[3]                // czwarta litera napisu, czyli a
napis.length            // długość napisu, czyli 4
napis.indexOf('b')      // pozycja pierwszego wystąpienia
                        // 'b' w napisie, czyli 1
napis.indexOf('e')      // 'e' nie występuje w napisie,
                        // więc zwracane jest -1
napis.lastIndexOf('a')  // ostatnie wystąpienie 'a'
                        // w napisie 3
napis.substr(1,2);      // od pozycji 1 (czyli drugiej
                        // litery) zwróć dwie litery,
                        // czyli 'bc'
```

Zadania do samodzielnego wykonania

1. Zmień program tak, aby sprawdzał, czy opis planszy zawiera jedynie cyfry.
2. Wydziel funkcję generującą pole `input`.
3. Zmień program tak, aby nie przechodził pętlą po napisie, ale po polach planszy (jak w przykładach w poprzednim punkcie).

Nadawanie elementom właściwości HTML

Nasza plansza nie wygląda ładnie. Na przykład nie ma na niej pionowych i poziomych kresek. Spróbujemy to zmienić. Biblioteka jQuery służy m.in. do nadawania elementom właściwości HTML. Dodajmy do programu funkcję `rysujKrawedzie` i zmodyfikujmy `$(document).ready` tak, aby z tej funkcji korzystał:

```
function rysujKrawedzie()
{
 var i;
 for(i = 0; i < 81; i++)
 {
  if ((i + 1) % 3 == 0)
   $('#td'+wspolrzedne(i)).css('border-right', '3px solid');
  if (i % 9 == 0)
   $('#td'+wspolrzedne(i)).css('border-left', '3px solid');
  if (Math.floor(i / 9) % 3 == 0)
   $('#td'+wspolrzedne(i)).css('border-top', '3px solid');
  if (i > 71)
   $('#td'+wspolrzedne(i)).css('border-bottom', '3px solid');
 }
}

$(document).ready(function(){
    $("#plansza").html(Plansza(opisPlanszy));
    rysujKrawedzie();
})
```

Widzimy teraz, że zmodyfikowaliśmy szerokości ramki i nasze Sudoku wygląda „ładniej”. Ale jak się nam to udało zrobić? Przyjrzyjmy się instrukcji:

```
$('#td'+wspolrzedne(i)).css('border-right', '3px solid');
```

`#td+wspolrzedne(i)` to napis identyfikujący jedną z komórek tablicy (wcześniej rozsądnie ponumerowaliśmy wszystkie komórki po kolei ustawiając im właściwość `id`), a `.css`, to funkcja, która ustawia właściwość CSS. W tym przypadku ustawiamy prawą ramkę. Inne ważne właściwości elementów HTML to:

- `background-color` – kolor tła; a lista kolorów: http://www.w3schools.com/css/css_colornames.asp
- kolory można podawać w formacie `#FFFFFF`
- `font-weight` – czy czcionka ma być pogrubiona (bold)
- `text-decoration` – możliwość podkreślenia
- `margin` – cztery (top, bottom, left, right) marginesy
- `border` – ramki (np. border-top: 2px solid red)
- `pudding` – odstęp od ramki
- `width, height` – szerokość i wysokość elementu

Jak widzimy na powyższym przykładzie, arkuszy styli CSS używa się nie tylko do utrzymywania spójnego stylu na wielu różnych stronach HTML, ale również do określania tego, w jaki sposób mają być prezentowane elementy strony HTML wtedy, gdy mamy do czynienia tylko z jedną stroną. Należy starać się oddzielać treść strony, w języku HTML od jej wyglądu, w postaci CSS.

Zadania do wykonania

1. Zmień kolor wyświetlanych liter na czerwony.
2. Zmień kolor ramek na ładniejszy (chyba, że lubisz czarny).
3. Może uda Ci się doprowadzić do tego, żeby tekst w polach do wpisywania był wyśrodkowany?

3.10 TABLICE

Zapewne będziemy chcieli umieć sprawdzić, czy w wierszach, kolumnach i kwadratach nie powtarza się dwa razy ta sama liczba. W tym celu watro mieć funkcje, które obliczają indeksy komórek w poszczególnych wierszach i kwadratach. Funkcje te mogłyby zwracać tablice (9-elementowe), zawierające te indeksy. Napiszmy je zatem i dołączmy je do programu, bo się za chwilę przydadzą:

```
function wiersz(i)
{
 if (i < 1 || i > 9) alert("wiersz: Zły wiersz");
 w = new Array();
 for(k = 10; k <= 90; k+=10)
  w.push(k+i);
 return w;
}

function kolumna(i)
{
 if (i < 1 || i > 9) alert("kolumna: Zły argument");
 w = new Array();
 for(k = 1; k <= 9; k++)
  w.push(i*10 + k);
 return w;
}

function kwadrat(i)
{
```

```
 if (i < 1 || i > 9) alert("kwadrat: Zły argument");
 w = new Array();
 x = ((i - 1) % 3) * 3 + 1;
 y = (Math.floor((i - 1) / 3)) * 3 + 1;
 for(dx = 0; dx <= 2; dx++)
  for(dy = 0; dy <= 2; dy++)
   w.push((x + dx) * 10 + dy + y);
 return w;
}
```

Każda z funkcji zwraca tablicę `w`. Tablicę tworzymy pisząc

- `new Array()` – tablica pusta, lub
- `new Array(rozmiar)` – tablica o początkowo ustawionym rozmiarze.

Tablice w JS są dynamiczne, czyli nie mają z góry określonego rozmiaru. Elementy dodajemy na koniec tablicy za pomocą instrukcji `push()`. Operacje na tablicach są następujące:

- `new Array(6)` – utworzenie tablicy o sześciu elementach;
- `a[3]` – czwarty element tablicy `a`;
- `a.length` – rozmiar tablicy `a`;
- `a.push('cd')` – dodanie `cd` na koniec tablicy `a`;
- `a.pop()` – usunięcie elementu z końca tablicy `a` i jego zwrócenie;
- `a.indexOf('ab')` – szukanie elementu `ab` w tablicy `a`;
- `a.splice(2,5)` – usunięcie pięciu elementów począwszy od trzeciego elementu z tablicy `a`;

Zadania do samodzielnego wykonania

1. Wszystkie zmienne w powyższych funkcjach są globalne. Popraw to.
2. Jak działa wybieranie k-tego kwadratu?

3.11 DYNAMICZNE NADAWANIE ELEMENTOM WŁAŚCIWOŚCI

Załóżmy, że chcielibyśmy mieć podświetlony wiersz i kolumnę, nad którą znajduje się wskaźnik myszy. Można to zrobić ustawiając odpowiednim elementom wartość `background-color` na zielony. Ale jak to zrobić? Z pomocą przychodzi nam znowu biblioteka jQuery. Nadaliśmy już wszystkim komórkom klasę `.field`. Teraz wystarczy, aby po znalezieniu się wskaźnika myszy nad jakimś elementem klasy `field` została podświetlona odpowiednia kolumna i odpowiedni wiersz. W tym celu należy przygotować funkcję `podswietl(element)`, która podświetla wiersz, kolumnę i kwadrat w którym znajduje się `element`. Zakładając, że mamy już przygotowane funkcje `podswietl` i `wygas`, wystarczy dołączyć ich obsługę do `$(document).ready`. Popatrzmy na nowy fragment programu.

```
function podswietlTablice(t)
{
 for(i = 0; i < 9; i++)
 {
  $('#td'+t[i]).css('background-color', 'green');
 }
}

function podswietl(i)
{
 podswietlTablice(wiersz(i % 10));
 podswietlTablice(kolumna(Math.floor(i/10)));
 podswietlTablice(kwadrat(1 + Math.floor((i-10)/30) + 3
  * Math.floor(((i % 10)-1)/3)));
}
```

```
function wygas()
{
 $(".field").css('background-color', 'transparent');
}

$(document).ready(function(){
    $("#plansza").html(Plansza(opisPlanszy));
    rysujKrawedzie();
    $(".field").mouseover(function(){
        podswietl($(this).attr('id').substr(2,2));
    });
    $(".field").mouseout(function(){
        wygas();
    });
})
```

Jak to działa?

1. Podświetlanie jest zrealizowane poprzez ustawianie odpowiedniej właściwości `css`.
2. Funkcje `mouseover` i `mouseout` są wykonywane w momencie najechania wskaźnika myszy na element tablicy i zjechania z niego.
3. `$(this)` zwraca element w kontekście którego jest wykonywana funkcja (w naszym przypadku najechał lub zjechał z niego wskaźnik myszy).
4. Bez funkcji `wygas` tablica po pewnym czasie cała stałaby się zielona.
5. Kolor tła `transparent` oznacza brak koloru, czyli pole jest przeźroczyste.

Zadania do samodzielnego wykonania

1. Niech kolumny i wiersze będą podświetlone innym kolorem niż kwadraty.
2. Program działałby bardziej intuicyjnie, gdyby podświetlane były te pola, które odnoszą się do wybranego elementu (mającego focus), a nie tego, nad którym jest wskaźnik myszy. Odpowiednie zdarzenia (zamiast `mouseover` i `mouseout`) nazywają się `focus` i `blur`.

3.12 REAGOWANIE NA AKCJE UŻYTKOWNIKA

Pozostało nam jeszcze sprawdzenie, czy Sudoku jest poprawnie rozwiązane. Rozwiązanie poprawne to takie, w którym w każdym wierszu, kolumnie i kwadracie jest dziewięć różnych liczb. Musimy jeszcze ustalić, w którym momencie powinniśmy sprawdzać rozwiązanie. Dodamy w tym celu do naszej strony link `sprawdz`, a jego zdarzenie `click` (czyli moment w którym na niego klikamy) podepniemy do funkcji, która sprawdza poprawność rozwiązania:

```
function poprawnaTablica(t)
{
 var o = ""
 var result = true;
 for(i = 0; i < 9; i++)
 {
  var v = $('#i' + t[i]).val();
  if (!(v >= '1' && v <= '9')) return false;
  if (o.indexOf(v) != -1) result = false;
  o += v;
 }
 return result;
}

function poprawneRozwiazanie()
{
```

```
 $("#debug").html("");
 var i;
 for(i = 1; i <= 9; i++)
 {
  if (!poprawnaTablica(wiersz(i))) return false;
  if (!poprawnaTablica(kolumna(i))) return false;
  if (!poprawnaTablica(kwadrat(i))) return false;
 }
 return true;
}

$(document).ready(function(){
    $("#plansza").html(Plansza(opisPlanszy));
    rysujKrawedzie();
    $(".field input").focus(function(){
        podswietl($(this).parent().attr('id').substr(2,2));
    });
    $(".field input").blur(function(){
        wygas();
    });
    $("#sprawdz").click(function(event){
        if (poprawneRozwiazanie())
            alert("Dobrze");
        else
            alert("Niedobrze");
    });
})

</script>
</head>
<body>
 <div id='plansza'>Miejsce na planszę</div>
 <a href="#" id="sprawdz">Sprawdz</a>
</body>
</html>
```

Tak przygotowany program powinien sprawdzać poprawność rozwiązania.

Zadania do samodzielnego rozwiązania

1. Spowoduj, aby program wypisywał, gdzie i dlaczego jest błąd w rozwiązaniu.

3.13 POMYSŁY NA DALSZE ROZSZERZENIA

Program można w tym momencie rozszerzać w wielu kierunkach. Wymienimy klika z nich:

1. To Sudoku nie wygląda ładnie. Popracuj nad jego wyglądem.
2. Przydatny byłby licznik czasu, najlepiej taki, który udostępniałby funkcję `STOP` (pauza).
3. Ciekawy byłby portal, na którym byłoby wiele Sudoku do rozwiązania.

LITERATURA

1. Crockford D., *JavaScript – mocne strony*, Helion, Gliwice 2009
2. Powers S., *JavaScript. Wprowadzenie*, Helion, Gliwice 2007

DODATEK. LISTING PROGRAMU

Oto pełny listing programu, który pisaliśmy:

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <title>Sudoku</title>
  <script type="text/javascript" src="jquery.js"></script>
  <script type="text/javascript">

var opisPlanszy =
"020730001" +
"009010047" +
"000208900" +
"000600802" +
"207853406" +
"804007000" +
"003405000" +
"640080700" +
"100072090";

function wspolrzedne(i)
{
 if (i < 0 || i > 80) alert("wspolrzedne: zle dane");
 return Math.floor(i/9) + 10 * (i % 9) + 11;
}

function Plansza(opis)
{
 var i;
 plansza = "<table>";
 for(i = 0; i < opis.length; i++)
 {
  if (i % 9 == 0) plansza += "<tr>";
  plansza += "<td class='field' id='td"+ wspolrzedne(i) +"'>";
  if (opis[i] == '0')
  {
   plansza += "<input type='text' size='1' maxlength='1' id='i"
    + wspolrzedne(i) + "'>";
  }
  else
  {
   plansza += "<input type='text' size='1' maxlength='1' readonly= '' id='i"
    + wspolrzedne(i) + "' value='"+opis[i]+"'>";
  }
  plansza += "</td>";
  if (i % 9 == 8) plansza += "</tr>";
 }
 return plansza+"</table>";
};

function rysujKrawedzie()
{
 var i;
 for(i = 0; i < 81; i++)
```

```
 {
  if ((i + 1) % 3 == 0)
   $('#td'+wspolrzedne(i)).css('border-right', '3px solid');
  if (i % 9 == 0)
   $('#td'+wspolrzedne(i)).css('border-left', '3px solid');
  if (Math.floor(i / 9) % 3 == 0)
   $('#td'+wspolrzedne(i)).css('border-top', '3px solid');
  if (i > 71)
   $('#td'+wspolrzedne(i)).css('border-bottom', '3px solid');
 }
}

function wiersz(i)
{
 if (i < 1 || i > 9) alert("wiersz: Zły wiersz");
 w = new Array();
 for(k = 10; k <= 90; k+=10)
  w.push(k+i);
 return w;
 }

function kolumna(i)
{
 if (i < 1 || i > 9) alert("kolumna: Zły argument");
 w = new Array();
 for(k = 1; k <= 9; k++)
  w.push(i*10 + k);
 return w;
}

function kwadrat(i)
{
 if (i < 1 || i > 9) alert("kwadrat: Zły argument");
 w = new Array();
 x = ((i - 1) % 3) * 3 + 1;
 y = (Math.floor((i - 1) / 3)) * 3 + 1;
 for(dx = 0; dx <= 2; dx++)
  for(dy = 0; dy <= 2; dy++)
   w.push((x + dx) * 10 + dy + y);
 return w;
}

function podswietlTablice(t)
{
 for(i = 0; i < 9; i++)
 {
  $('#td'+t[i]).css('background-color', 'green');
 }
}

function podswietl(i)
{
 podswietlTablice(wiersz(i % 10));
 podswietlTablice(kolumna(Math.floor(i/10)));
```

```
 podswietlTablice(kwadrat(1 + Math.floor((i-10)/30) + 3
  * Math.floor(((i % 10)-1)/3)));
}

function wygas()
{
 $(".field").css('background-color', 'transparent');
}

function poprawnaTablica(t)
{
 var o = ""
 var result = true;
 for(i = 0; i < 9; i++)
 {
  var v = $('#i' + t[i]).val();
  if (!(v >= '1' && v <= '9')) return false;
  if (o.indexOf(v) != -1) result = false;
  o += v;
 }
 return result;
}

function poprawneRozwiazanie()
{
 $("#debug").html("");
 var i;
 for(i = 1; i <= 9; i++)
 {
  if (!poprawnaTablica(wiersz(i))) return false;
  if (!poprawnaTablica(kolumna(i))) return false;
  if (!poprawnaTablica(kwadrat(i))) return false;
}
 return true;
}

$(document).ready(function(){
    $("#plansza").html(Plansza(opisPlanszy));
    rysujKrawedzie();
    $(".field input").focus(function(){
        podswietl($(this).parent().attr('id').substr(2,2));
    });
    $(".field input").blur(function(){
        wygas();
    });
    $("#sprawdz").click(function(event){
        if (poprawneRozwiazanie())
            alert("Dobrze");
        else
            alert("Niedobrze");
    });
})

  </script>
```

```
 </head>
 <body>
  <div id='plansza'>Miejsce na planszę</div>
  <a href="#" id="sprawdz">Sprawdz</a>
 </body>
</html>
```

W projekcie **Informatyka +**, poza wykładami i warsztatami, przewidziano następujące działania:

- 24-godzinne kursy dla uczniów w ramach modułów tematycznych
- 24-godzinne kursy metodyczne dla nauczycieli, przygotowujące do pracy z uczniem zdolnym
- nagrania 60 wykładów informatycznych, prowadzonych przez wybitnych specjalistów i nauczycieli akademickich
- konkursy dla uczniów, trzy w ciągu roku
- udział uczniów w pracach kół naukowych
- udział uczniów w konferencjach naukowych
- obozy wypoczynkowo-naukowe.

Szczegółowe informacje znajdują się na stronie projektu

www.informatykaplus.edu.pl