



Zaawansowane Techniki WWW (HTML, CSS i JavaScript)

Dr inż. Marcin Zieliński

Środa 15:30 - 17:00 sala: A-1-04

WYKŁAD 5

Wykład dla kierunku: **Informatyka Stosowana II rok**

Rok akademicki: **2015/2016 - semestr zimowy**

Przypomnienie z poprzedniego wykładu

Strony na urządzeniach mobilnych.

Responsive Web Design (RWD)

Metodologia Mobile-First

CSS3 i Mediaqueries

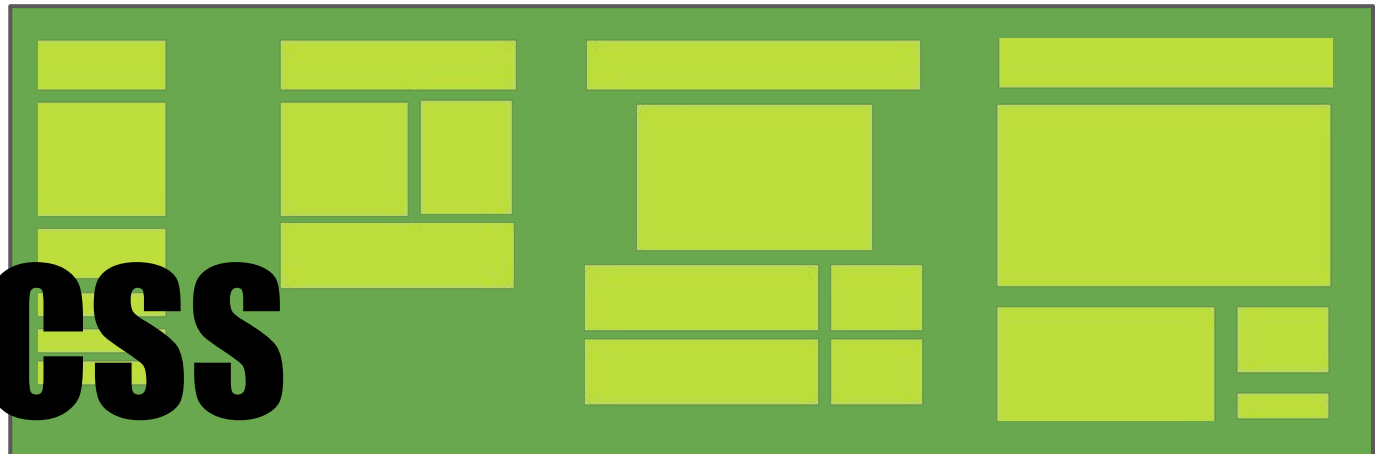
Jak stosować Mediaqueries?



ZASADA:

W ogólności zasada jaka powinna przyświecać tworzeniu stron zgodnie z metodologią RWD jest tworzenie jednego pliku HTML, a dla formatowania jego wyglądu wiele “layoutów” w CSS które będą zawierać odpowiednie reguły wyświetlania strony w zależności od rozdzielczości ekranu.

n x CSS

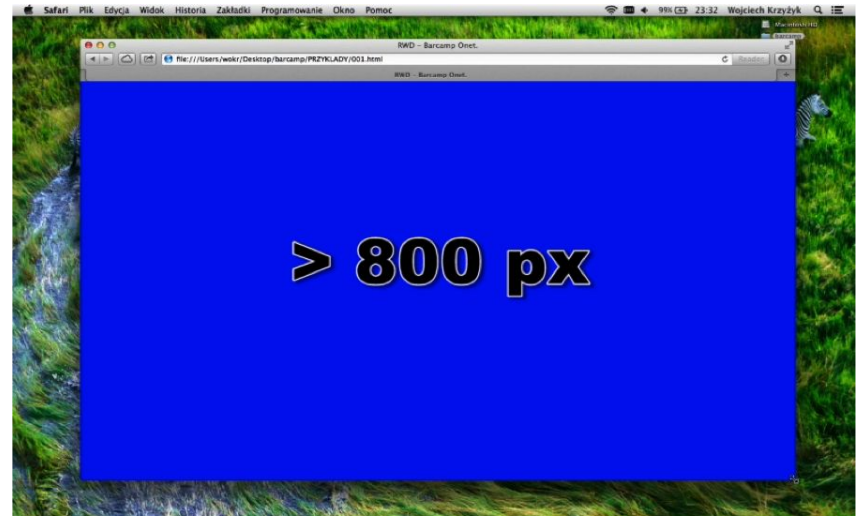


CSS3 i Mediaqueries

Dodatkowo do poza określaniem konkretnej wartości atrybutów można zadać zakresy dolny i górny:

Reguły ograniczające
z góry i z dołu:

{ min-
max- }



```
@media screen and ( min-width:  
max-width
```

CSS3 i Mediaqueries

Przykład zastosowania:

```
<style>
  @media all and (min-width:480px) and (max-width:800px) {
    div {
      background-color:green;
    }
  }
</style>
```

Definicja określona powyżej mówi przeglądarce że tło elementu <div> ma zostać ustawione na kolor zielony na wszystkich typach urządzeń tylko wtedy kiedy szerokość okna przeglądarki będzie zawierać się w przedziale od 480px od 800px.

CSS3 i Mediaqueries

Przykład zastosowania:

```
<style>
  @media all and (min-color-index:256) {
    div {
      background-color:green;
    }
  }
</style>
```

W przypadku tej reguły kolor tła elementu <div> zostanie zmieniony na zielony tylko wtedy kiedy wartość głębi kolorów możliwych do wyświetlenia na danym urządzeniu nie jest mniejsza od 256.

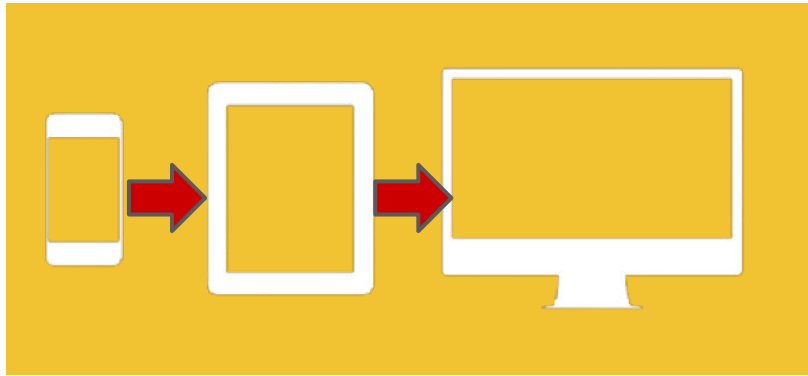
CSS3 i Mediaqueries

Zastosowanie operatora “only” nie ma on wpływu na definicję reguły, ale zapewnia kompatybilność ze starszymi przeglądarkami, które nie obsługują MediaQueries. Operator był dostępny w wersji 2.1 CSS - ograniczał reguły css dla danego typu "media". Ponieważ starsze przeglądarki nie rozpoznają reguł - nie zinterpretują prawidłowo cssów z media.

```
<style>
  @media only screen (min-width:480px) and (max-width:800px) {
    div {
      background-color:red;
    }
  }
</style>
```

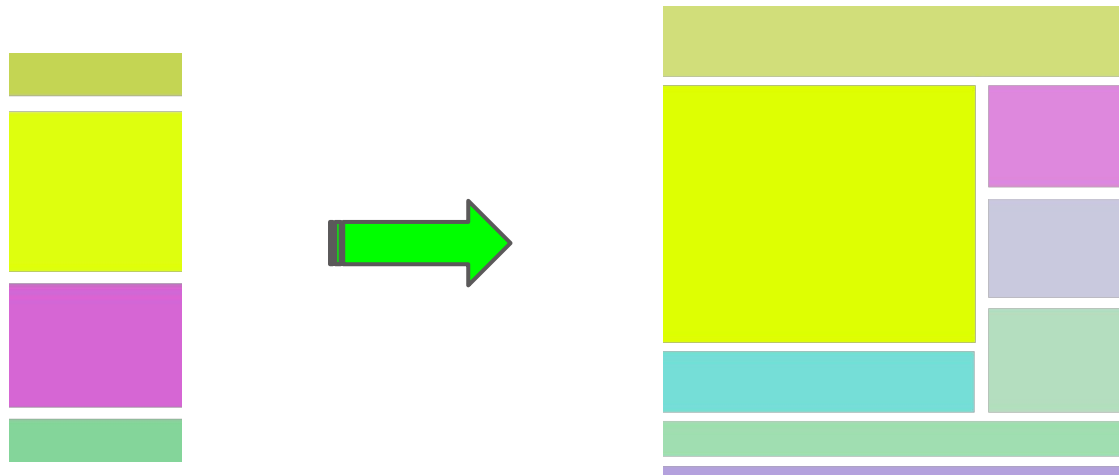
Które CSSy tworzyć najpierw ?

Które CSSy tworzyć najpierw ?



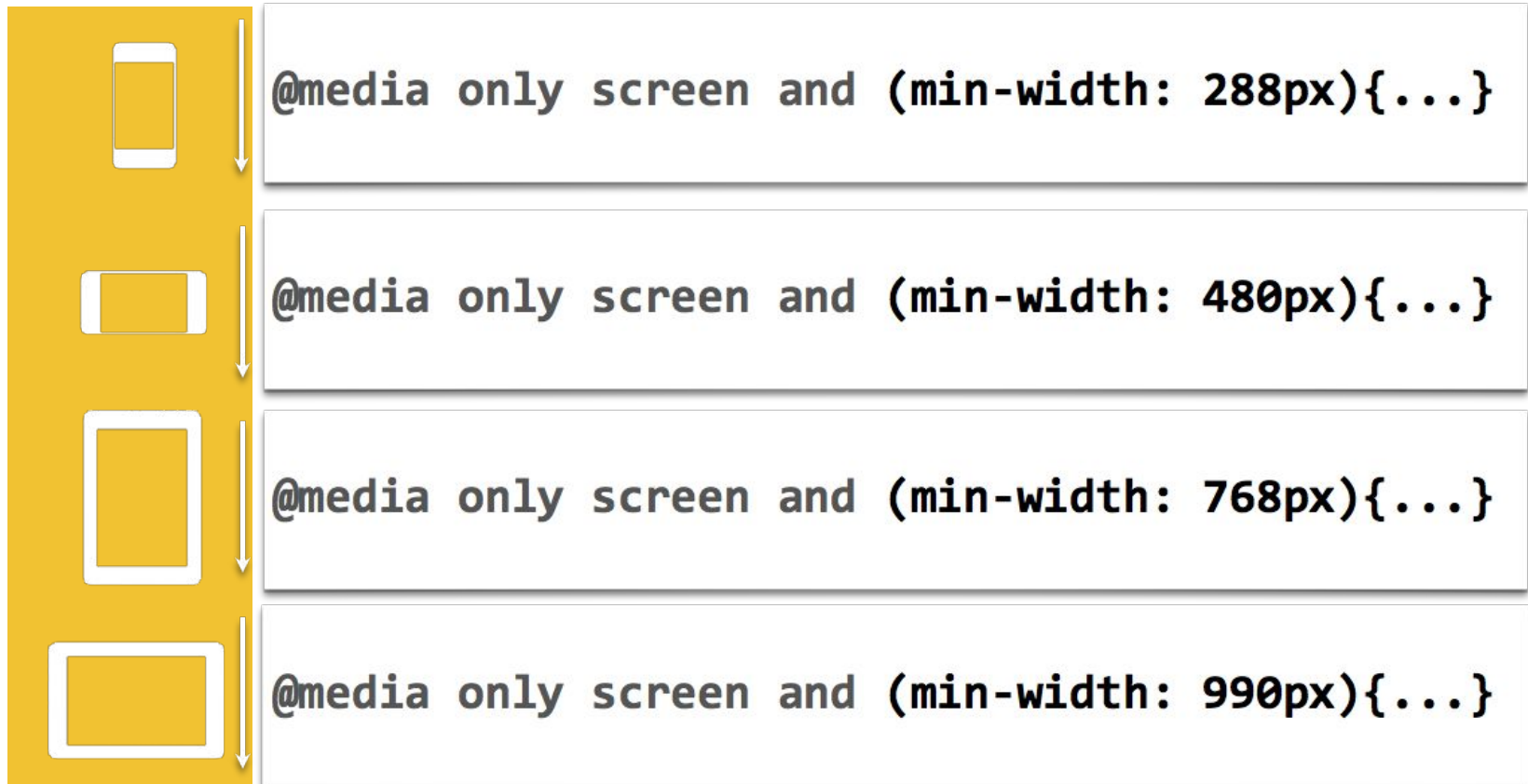
**Filozofia
Mobile-First**

Zgodnie z filozofią MobileFirst tworzenie strony powinno być rozpoczęte od najmniejszych ekranów czyli dla urządzeń mobilnych.

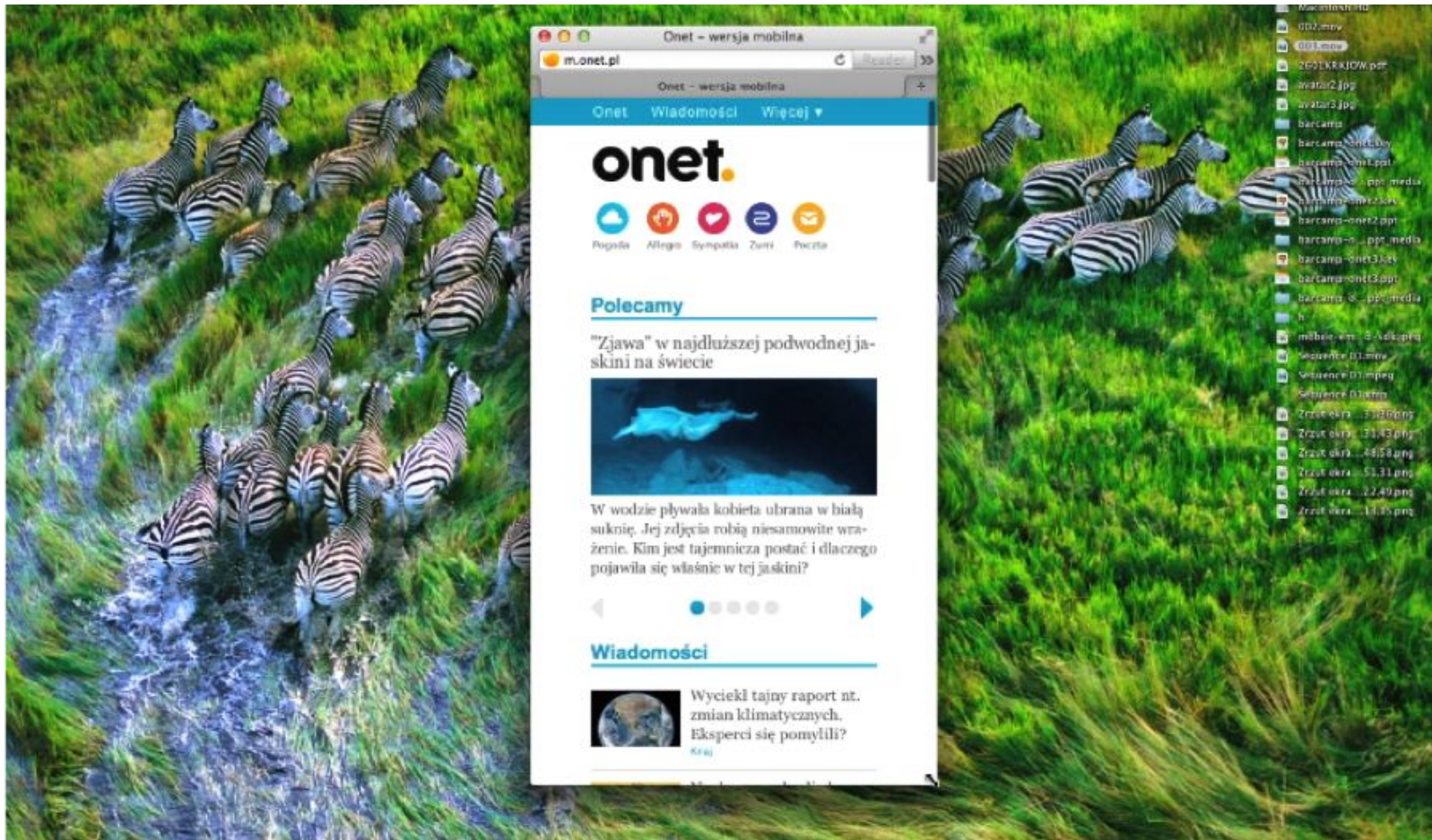


Mobile First

Zgodnie z tymi zasadami CSSy powinniśmy układać mniej więcej wg takiego schematu:



Mobile First

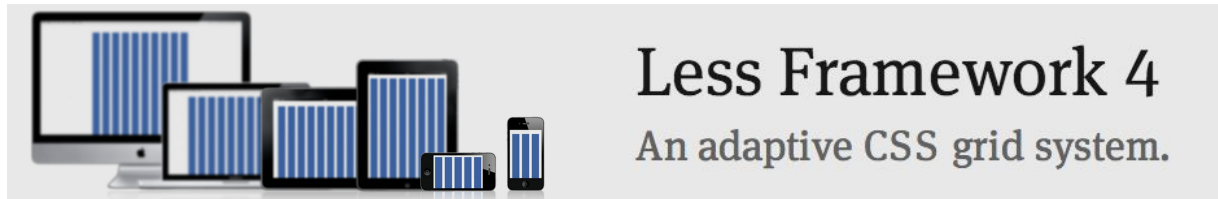


Narzędzia ułatwiające pracę

Pracę z Mediaqueries oraz z projektowaniem widoków ułatwiają frameworki m.in. Lessframework, Cssgrid, Columnal czy Foundation.

Narzędzia ułatwiające pracę

Pracę z Mediaqueries oraz z projektowaniem widoków ułatwiają frameworki m.in. Lessframework, Cssgrid, Columnal czy Foundation.



<http://lessframework.com/>



<http://www.columnal.com/>



<http://foundation.zurb.com/>



<http://www.1140px.com/>

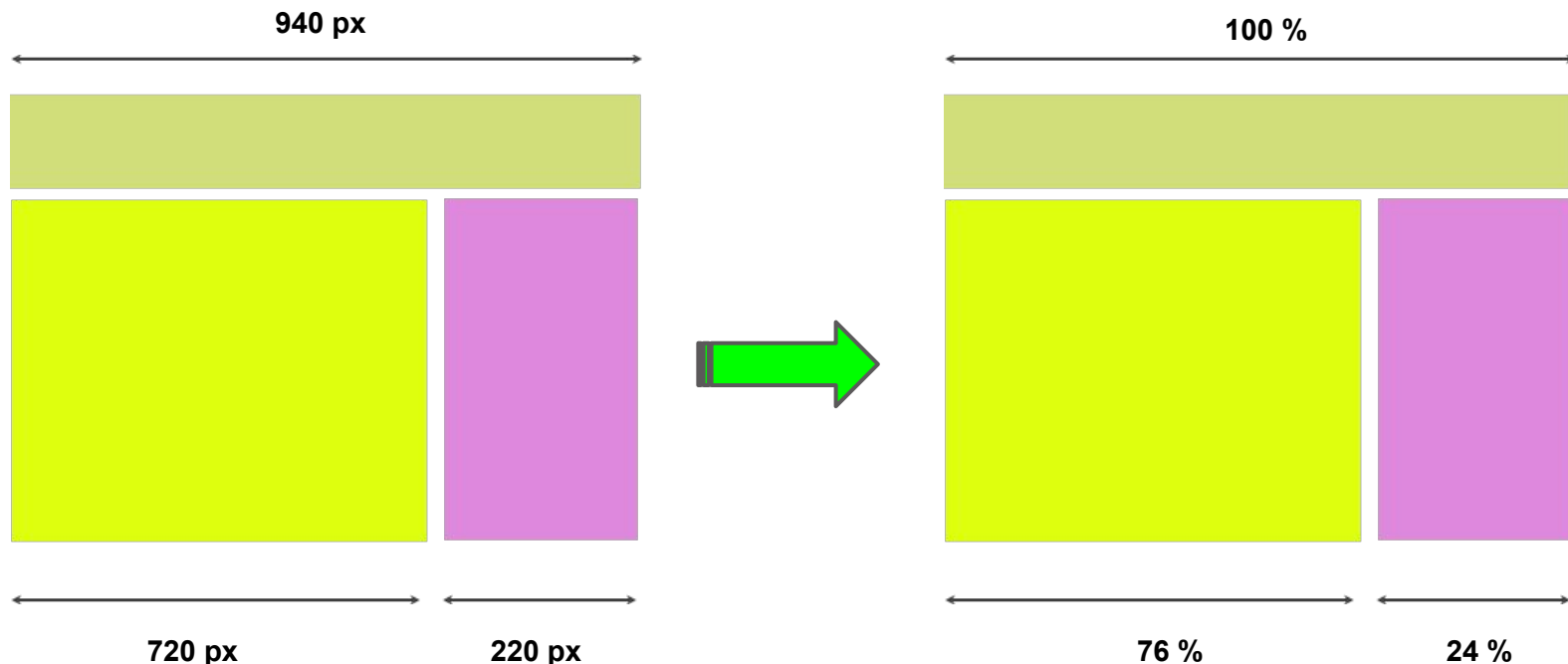
Frameworki te dostarczają zdefiniowanych reguł CSS/Mediaqueries dla zdefiniowanych szerokości ekranów. Oczywiście domyślne wartości możemy dowolnie modyfikować. Ułatwiają nam budowanie layoutów responsywnych stron, a niektóre jak np Foundation oferują już gotowe prototypy.

Nowe podejście do definiowania rozmiarów elementów

Chcąc zaprojektować "elastyczną" stronę dopasowującą się do całego ekranu powinniśmy zrezygnować z jednostek bezwzględnych wyrażanych np. w px i zastąpić je jednostkami względnymi w procentach %.

Pozwoli to uzyskać dopasowany układ do różnych szerokości ekranów.

Atrybuty max-width powinny być używane często zwłaszcza w przypadku elementów, które mogą nam "rozpychać layout" tj. obrazków, osadzonych odtwarzaczy video etc.



Funkcja *calc()*

CSS został wyposażony w specjalną funkcję o nazwie “*calc()*”, która pozwala wykonywać nieskomplikowane obliczenia matematyczne do określenia wartości danej cechy jeśli jest ona liczbą. Do dyspozycji mamy następujące działania: +, - , * (mnożenie), / (dzielenie).

Z wykorzystaniem tej funkcji możemy określić za pomocą równania matematycznego jak ma się dopasować do ekranu dana cecha.

Funkcja *calc()*

CSS został wyposażony w specjalną funkcję o nazwie “*calc()*”, która pozwala wykonywać nieskomplikowane obliczenia matematyczne do określenia wartości danej cechy jeśli jest ona liczbą. Do dyspozycji mamy następujące działania: +, -, * (mnożenie), / (dzielenie).

Z wykorzystaniem tej funkcji możemy określić za pomocą równania matematycznego jak ma się dopasować do ekranu dana cecha.

```
<style>
  @media only screen (max-width:800px){
    .calc {
      /* Firefox */
      width: -moz-calc(75% - 100px);
      /* WebKit: Safari, Chrome */
      width: -webkit-calc(75% - 100px);
      /* Opera */
      width: -o-calc(75% - 100px);
      /* Standard */
      width: calc(75% - 100px);
    }
  }
</style>
```


Funkcja *calc()*

Zgodność z przeglądarkami:

calc() as CSS unit value  - CR

Method of allowing calculated values for length units, i.e. width:

calc(100% - 3em)

Global 77.04% + 5.27% = 82.32%
 unprefixed: 75.83% + 5.27% = 81.1%

Current aligned Usage relative Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		33						
8		35	5.1				4.1	
9	31	36	7		7.1		4.3	
10	32	37	7.1		8		4.4	
11	33	38	8	25	8.1	8	4.4.4	38
	34	39		26			37	
	35	40		27				
	36	41						

(dane ze strony <http://caniuse.com/#feat=calc>)

Jak testować Mediaqueries ?

Jak sprawdzić czy przygotowana strona działa wszędzie poprawnie??

Na rynku mamy niezliczoną liczbę urządzeń, mnogość systemów operacyjnych, na każdym jest po kilka różnych przeglądarek. Nie mamy dostępu do wszystkich tych urządzeń, a problemów jakich możemy się spodziewać jest cały szereg: brak obsługi Javascriptu, brak obsługi Mediaqueries, problemy z layoutem etc...



Jak testować Mediaqueries ?

Jak sprawdzić czy przygotowana strona działa wszędzie poprawnie??

Na rynku mamy niezliczoną liczbę urządzeń, mnogość systemów operacyjnych, na każdym jest po kilka różnych przeglądarek. Nie mamy dostępu do wszystkich tych urządzeń, a problemów jakich możemy się spodziewać jest cały szereg: brak obsługi Javascriptu, brak obsługi Mediaqueries, problemy z layoutem etc...



 Windows Phone

XPERIA
Sony Smartphone

Jak poradzić sobie testowaniem ?

ZTE中兴



Samsung
GALAXY S III

Opera Mobile

Jak testować Mediaqueries ?

Metoda nr 1

Skalowanie okna przeglądarki

Jak testować Mediaqueries ?

Metoda nr 1

Skalowanie okna przeglądarki

Metoda nr 2

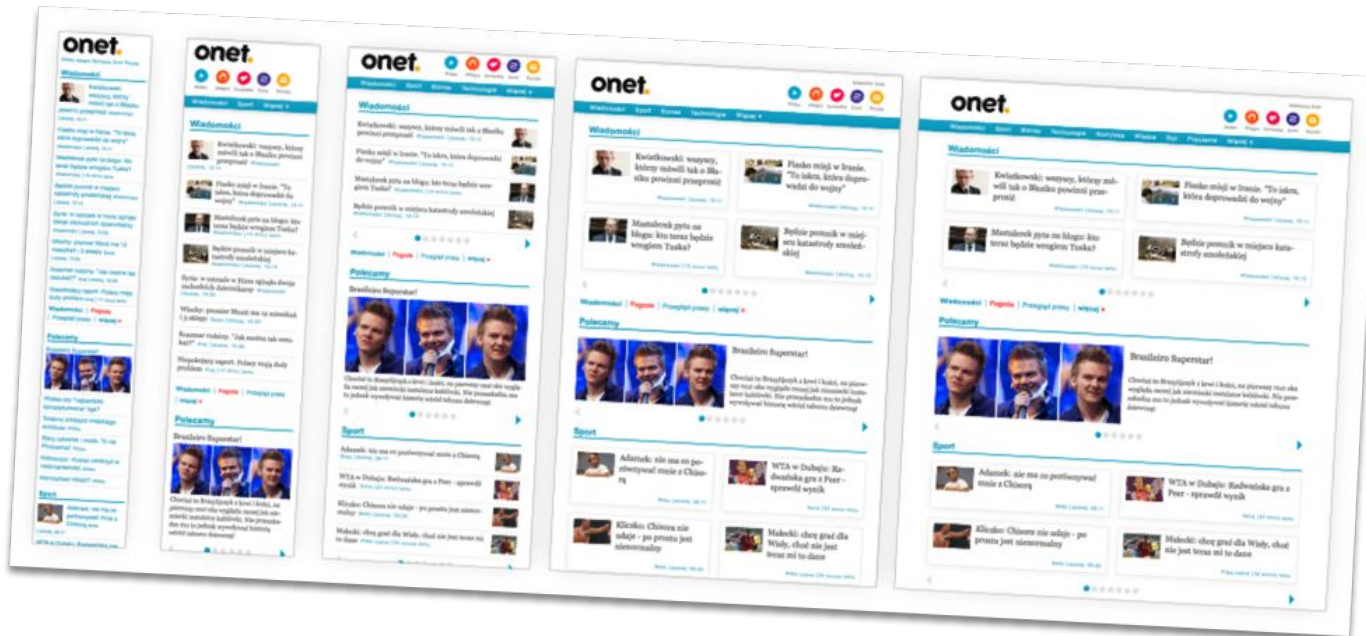
**Narzędzia deweloperski w
przeglądarkach**

Jak testować Mediaqueries ?

Metoda nr 1

Skalowanie okna przeglądarki

Zawsze najlepiej zacząć od najprostszej metody - skalowania okna przeglądarki. Wydaje się banalne, ale jest to najszybsza metoda wyłapania wszystkich niedoskonałości.

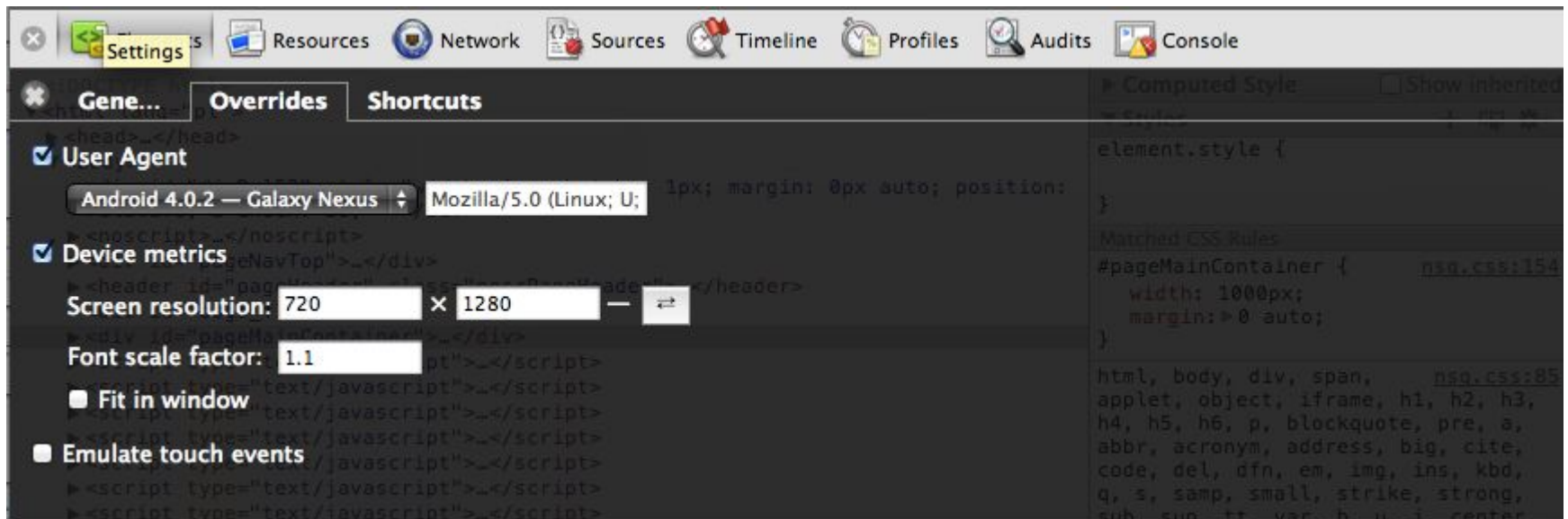


Jak testować Mediaqueries ?

Metoda nr 2

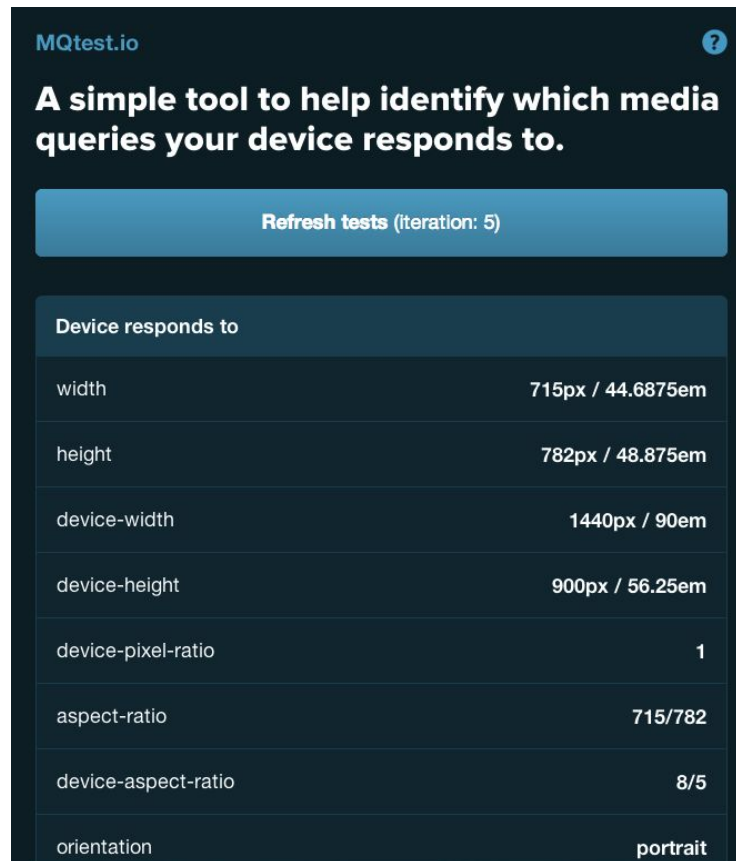
Narzędzia deweloperski w przeglądarkach

Wiele nowoczesnych przeglądarek posiada specjalne rozszerzenia pozwalające używać tzw. Trybu Debugowania, pozwalającego też na zasymulować różnych wymiarów ekranu.



Jak testować Mediaqueries ?

Dodatkowo można również za pomocą strony www.mqtest.io poznać wszelkie informacje o regułach jakie są obsługiwane na urządzeniu z którego ją odwiedziliśmy, a które możemy wykorzystać w mediaqueries.




MQtest.io ?

A simple tool to help identify which media queries your device responds to.

Refresh tests (iteration: 5)

Device responds to	
width	715px / 44.6875em
height	782px / 48.875em
device-width	1440px / 90em
device-height	900px / 56.25em
device-pixel-ratio	1
aspect-ratio	715/782
device-aspect-ratio	8/5
orientation	portrait

Wstęp do JavaScript



JS

Wstęp do JavaScript

W latach 1994 - 1995 podstawy języka JavaScript zostały stworzone przez firmę Netscape, a następnie w kolejnym etapie był rozwijany we współpracy z firmą SUN. Twórcą standardu jest Brendan Eich. Język ten pozwala na rozszerzenie funkcji oferowanych przez HTML oraz CSS w celu zwiększenia funkcjonalności stron internetowych.

JavaScript posiada wszystkie podstawowe elementy poprawnego języka programowania: zmienne, instrukcje warunkowe, pętle, instrukcje wejścia/wyjścia, tablice, funkcje, a zwłaszcza obiekty. Język ten jest oparty na obiektach (ang. *object-based*) i jest sterowany zdarzeniami (ang. *event-driven*).



Wstęp do JavaScript

Cechy języka JavaScript:

- Zapewnia obsługę DOM (Document Object Model),
- Brak statycznej kontroli typów zmiennych (trudności z wykryciem błędów).
- Współpraca z formatem JSON,
- Wbudowane dziedziczenie prototypowe,
- Brak klas typowych z innych języków programowania.

Opis standardu języka:

<http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>

Obecnie najnowszą rekomendowaną wersją JavaScript jest wersja 1.8.5.

Jak stosować kod w języku JS ?

Załączenie do strony internetowej kodu napisanego w języku JavaScript (w skrócie skryptu) można dokonać na dwa sposoby:

- osadzić skrypt w kodzie HTML strony (embedded-script),
- umieścić w osobnym pliku (linked-script).

HTML5

Embedded-Script

```
<body>
  <script>
    alert("Hello World!");
  </script>
</body>
```

Linked-Script

```
<head>
  <script src="skrypt.js">
  </script>
</head>
```

Plik
.html

```
alert("Hello World!");
```

Plik
.js

Jak stosować kod w języku JS ?

Załączenie do strony internetowej kodu napisanego w języku JavaScript (w skrócie skryptu) można dokonać na dwa sposoby:

- osadzić skrypt w kodzie HTML strony (embedded-script),
- umieścić w osobnym pliku (linked-script).

HTML4 /
XHTML

Embedded-Script

```
<body>
  <script type="text/javascript">
    alert("Hello World!");
  </script>
</body>
```

Linked-Script

```
<head>
  <script
    type="text/javascript"
    src="skrypt.js">
  </script>
</head>
```

Plik
.html

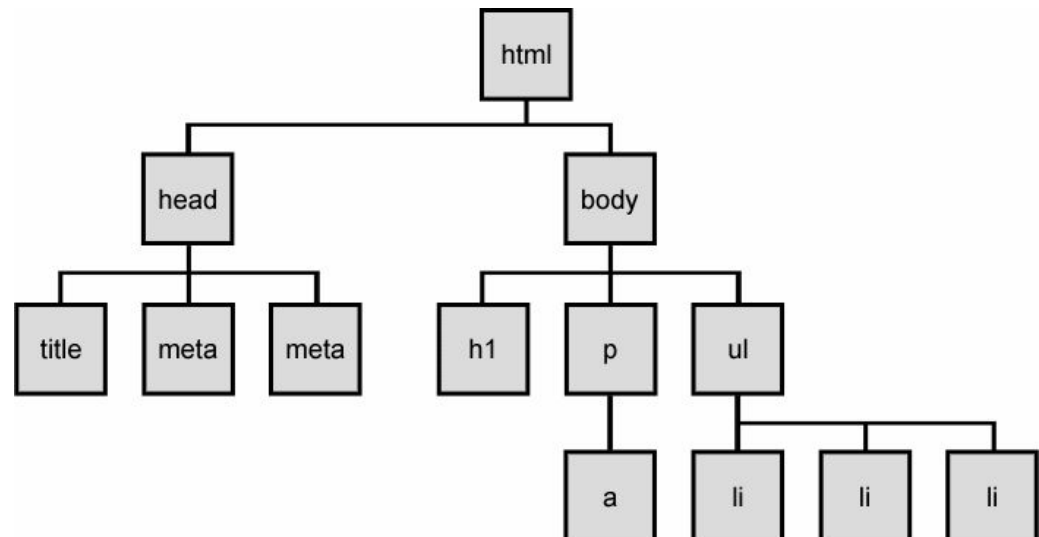
```
alert("Hello World!");
```

Plik
.js

DOM (Document Object Model)

DOM (Document Object Model) [*Obiektowy Model Dokumentu*] - jest to model reprezentujący budowę dokumentów XML, HTML w postaci hierarchicznej struktury obiektów i elementów. Model DOM jest niezależny od platformy i środowiska programistycznego. Pozwala na dostęp do elementów dokumentu (jego struktury) oraz wykonywanie czynności jak dodawanie, usuwanie i modyfikacja zawartości poszczególnych elementów.

Przykład modelu DOM dla prostego dokumentu hipertekstowego



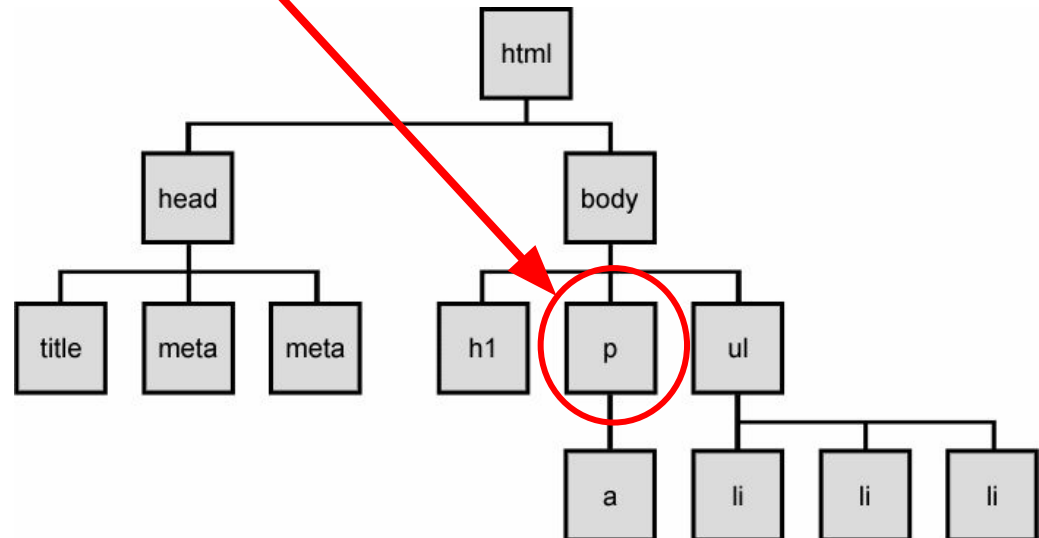
DOM (Document Object Model)

Dzięki zdefiniowanym w modelu metodom i klasą możliwy jest dostęp do dowolnego elementu dokumentu tzw. węzła (node), np.:

```
document.getElementsByTagName("p") ;
```

Przykład modelu DOM dla prostego dokumentu hipertekstowego

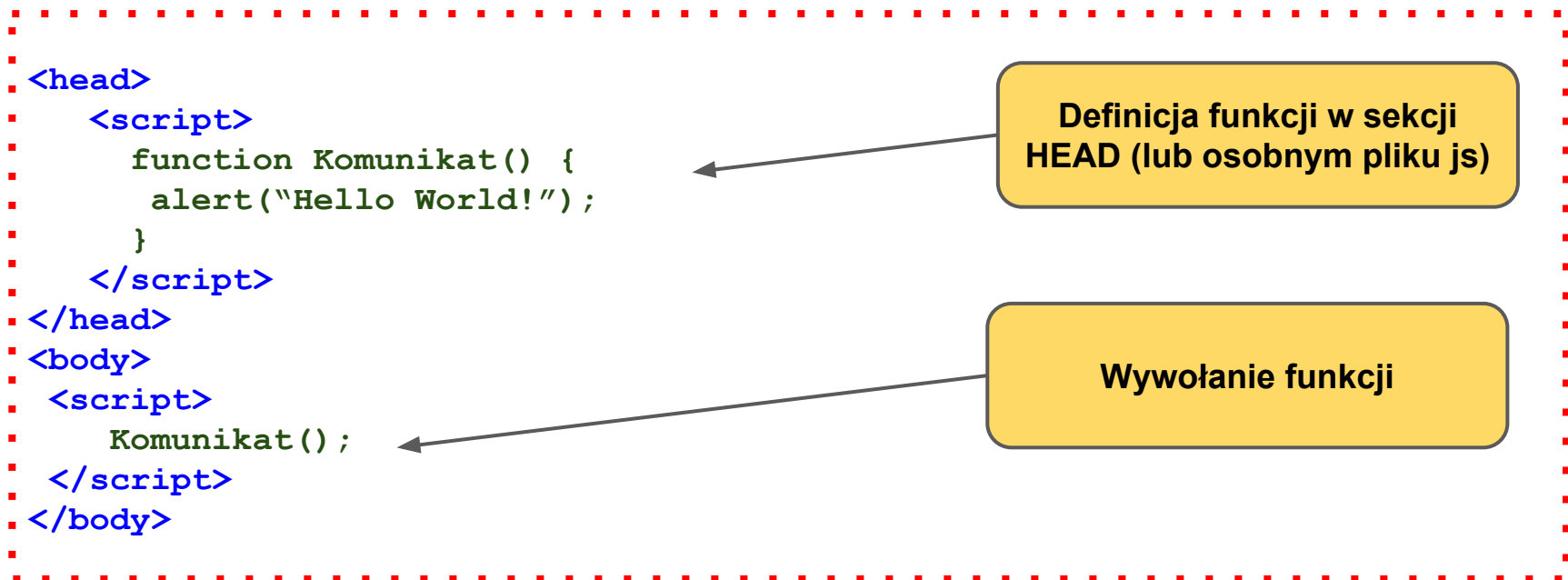
<http://www.w3.org/DOM/>



Jak stosować kod w języku JS ?

Dobre praktyki

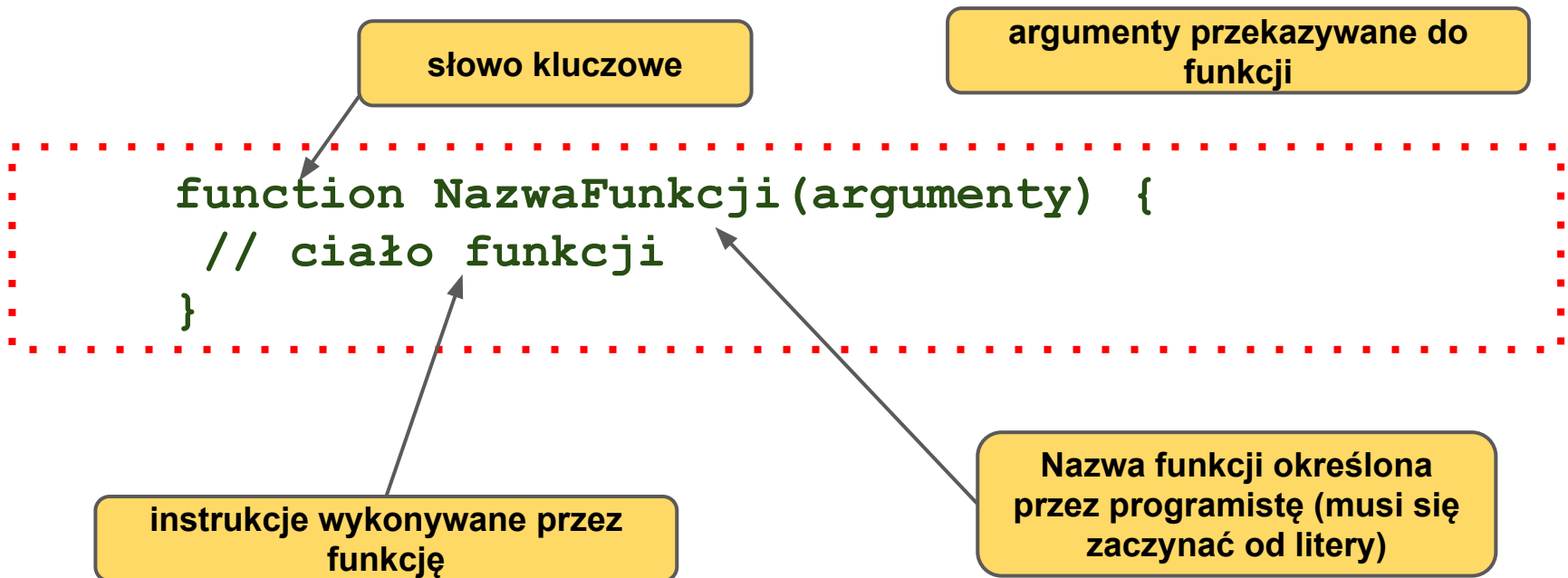
W poprzednich przykładach, które pokazywały sposoby osadzenia kodu używaliśmy funkcji wbudowanej `alert()`, która umożliwia wyświetlanie komunikatów na stronie w formie wyskakującego okienka. Jednak taki zapis powoduje znów mieszanie kodu HTML i JS, dlatego zalecaną metodą jest stosowanie funkcji definiowanych przez użytkownika:



Zaczniemy od funkcji ...

Zaczniemy od zdefiniowania (nawet) jak tworzyć funkcje, które są podstawowymi narzędziami i jednostkami modularnymi wykorzystywanymi przez programistę JavaScriptu:

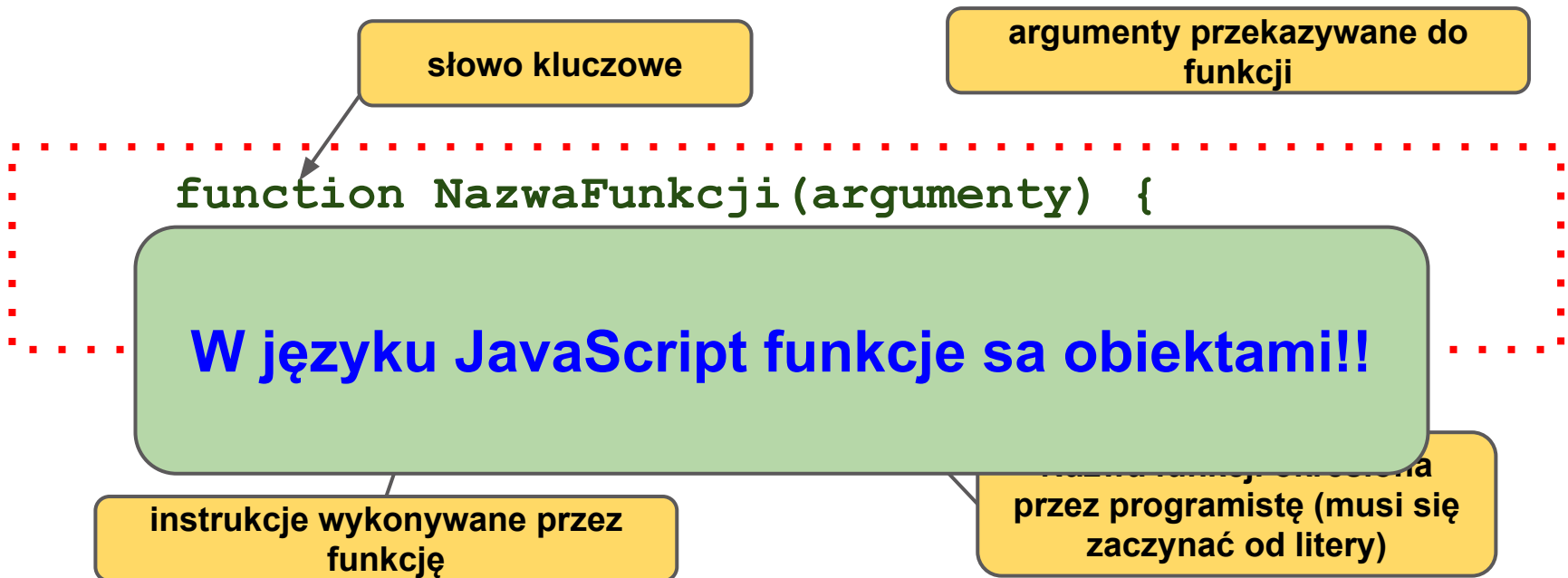
Funkcja nazwana:



Zaczniemy od funkcji ...

Zaczniemy od zdefiniowania (nawet) jak tworzyć funkcje, które są podstawowymi narzędziami i jednostkami modularnymi wykorzystywanymi przez programistę JavaScriptu:

Funkcja nazwana:



Kiedy przeglądarka nie obsługuje JavaScriptu

W przypadku kiedy przeglądarka nie obsługuje języka JavaScript lub jest on celowo wyłączony przez użytkownika, operacje zawarte w skrypcie nie zostaną wykonane. Dlatego istotnym jest poinformowanie użytkownika że jego przeglądarka nie wspiera obsługi JavaScript. Można to uczynić stosując specjalny znacznik `<noscript>`, za pomocą którego możemy użytkownikowi wystawić jasny komunikat o zaistniałej sytuacji.

Kiedy przeglądarka nie obsługuje JavaScriptu

W przypadku kiedy przeglądarka nie obsługuje języka JavaScript lub jest on celowo wyłączony przez użytkownika, operacje zawarte w skrypcie nie zostaną wykonane. Dlatego istotnym jest poinformowanie użytkownika że jego przeglądarka nie wspiera obsługi JavaScript. Można to uczynić stosując specjalny znacznik `<noscript>`, za pomocą którego możemy użytkownikowi wystawić jasny komunikat o zaistniałej sytuacji.

```
<script type="text/javascript">
document.write("Witaj świecie!")
</script>
<noscript>
Twoja przeglądarka nie obsługuje języka Javascript.
</noscript>
```

Znacznik ten definiuje treść alternatywną, która będzie wyświetlona, gdy przeglądarka nie obsługuje języka **JavaScript**.

Zmienne

JavaScript jest językiem który nie obsługuje statycznych typów zmiennych, a więc nie możemy zadeklarować że dana zmienna jest np. typu całkowitego, znakowego lub zmiennoprzecinkowego. Może to powodować liczne błędy ale jako że JS nie jest językiem kompilowalnym nie powoduje tak dużych trudności. W języku JavaScript zmienne deklaruje się przez słowo kluczowe “var”:

```
var moja_zmienna_1; // brak typu  
var moja_zmienna_2 = 2; // typ całkowity  
var moja_zmienna_3 = 'a'; // typ znakowy  
var moja_zmienna_4 = 'Jan'; // typ łańcuchowy  
var moja_zmienna_5 = '3.1415'; //
```

Typ zmiennej jest ustalany w momencie przypisania wartości.

Zmienne

Natomiast nie stoi na przeszkodzie aby zrobić tak:

```
// definiujemy zmienną bez typu  
var zmienna;  
zmienna = 2; //ustalamy typ na całkowity  
zmienna = 3.4 // typ zmiennoprzecinkowy  
zmienna = 'Jan'; // typ łańcuchowy
```

W dowolnym momencie kodu możemy zmienić typ zmiennej przez przypisanie wartości!! Może to powodować pewne niebezpieczne sytuacje, dlatego należy bardzo rozważnie przypisywać wartości zmiennym.

Zakresy zmiennych

Typowo dla każdego języka programowania zmienne dzielimy na globalne i lokalne. Zmienne globalne oddziałują w obszarze całego skryptu JS i z dowolnego miejsca mamy do nich dostęp. Natomiast zmienne lokalne działają np. tylko w obrębie jednej funkcji:

```
var zmienna_globalna = 5;
function zrobCos() {
    var zmienna_lokalna = 7;
}
alert(zmienna_globalna);
alert(zmienna_lokalna); // to nie zadziała!!!
```

Zakresy zmiennych

Typowo dla każdego języka programowania zmienne dzielimy na globalne i lokalne. Zmienne globalne oddziałują w obszarze całego skryptu JS i z dowolnego miejsca mamy do nich dostęp. Natomiast zmienne lokalne działają np. tylko w obrębie jednej funkcji:

```
var zmienna_globalna = 5;
function zrobCos() {
    var zmienna_lokalna = 7;
}
alert(zmienna_globalna);
alert(zmienna_lokalna); // to nie zadziała!!!
```

Uwaga:

W środowisku programistów JS obowiązuje zasada, że należy używać zmiennych lokalnych (jak najczęściej funkcji), chyba że z pewnych względów ta sama zmienna musi być wykorzystana w wielu funkcjach wtedy może być globalna. Wynika to z tego że jeśli będzie za dużo zmiennych globalnych kod staje się nieczytelny i trudno jest je zorganizować oraz mamy ograniczoną przestrzeń nazw.

Operacje na zmiennych

Typowym działaniem w JS podobnie jak w innych językach programowania jest wykonywanie operacji na zmiennych. Można do tego celu użyć operatorów artmetycznych: dodawania, odejmowania, mnożenia, dzielenia.

```
var a = 5;  
var b = 6;  
var c = 9;  
var wynik = (a + b) / (c + 1);
```

Operacje na zmiennych

Typowym działaniem w JS podobnie jak w innych językach programowania jest wykonywanie operacji na zmiennych. Można do tego celu użyć operatorów artmetycznych: dodawania, odejmowania, mnożenia, dzielenia.

```
var a = 5;  
var b = 6;  
var c = 9;  
var wynik = (a + b) / (c + 1);
```

Operacji można używać również do łączenia łańcuchów w jeden ciąg:

```
var imie = "Marcin";  
var nazwisko = "Zielinski";  
var imieInazwisko = imie + " " + nazwisko;
```

Operatory logiczne

W języku JavaScript wszystkie instrukcje warunkowe zaczynają się od warunku, czyli wyrażenia które może zwrócić wartość logiczną prawda lub fausz. Aby utworzyć warunek można posłużyć się operatorami logicznymi:

Operator	Opis
==	jest równe
!=	nie jest równe
>	jest większe
<	jest mniejsze
>=	większe lub równe
<=	mniejsze lub równe
&&	i
	lub
!	zaprzeczenie

Instrukcje warunkowe

Najprostszą i jednocześnie najpopularniejszą instrukcją warunkową jest instrukcja “if”, która po obliczeniu wartości wyrażenia wykonuje dany ciąg instrukcji lub nie:

```
var liczba = 7;  
if( liczba < 6) { // warunek niespełniony  
    alert(liczba);  
}
```

```
var liczba = 7;  
if( liczba > 6) { // warunek spełniony  
    alert(liczba);  
}
```

```
var liczba = 7;  
if( liczba > 6) alert(liczba);
```

Instrukcje warunkowe

Rozbudowana wersja instrukcji “if” z wieloma warunkami:

```
var liczba = 7;
if( liczba < 6) {
    alert("Liczba jest mniejsza od 6");
}
else if (liczba >=6 && liczba < 100) {
    alert("liczba z zakresu 6 do 99);
}
else {
    alert("liczba poza zakresem");
}
```

Pętle

Podobnie jak w językach kompilowalnych do wykonania wielokrotnie tej samej czynności można zastosować pętle. W JavaScript pętla ma postać identyczną jak w języku C++:

```
for( var i=0; i < 5; i++) {  
    alert("To jest komunikat nr" + i);  
}
```

Wynik:

```
To jest komunikat nr 0  
To jest komunikat nr 1  
To jest komunikat nr 2  
To jest komunikat nr 3  
To jest komunikat nr 4
```

DOM (Document Object Model)

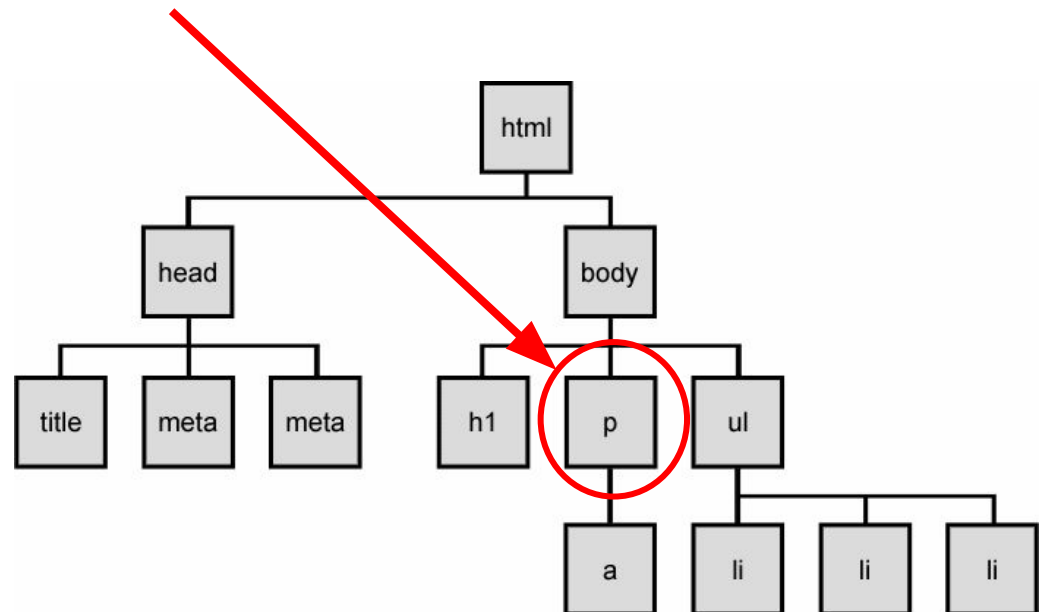
Elementy dokumentu HTML tworzą strukturę drzewiastą w której każdy znacznik pełni rolę węzła (node).

Dzięki zdefiniowanym w modelu DOM metodom i klasą możliwy jest dostęp do dowolnego węzła:

```
document.getElementsByTagName("p") ;
```

Przykład modelu DOM dla prostego dokumentu hipertekstowego

<http://www.w3.org/DOM/>



Tablice

Tablice są bardzo wygodne do przechowywania różnego rodzaju danych.

W JavaScript mamy pełną swobodę tworzenia tablic:

```
var kolory= [];  
kolory.push("zielony");  
kolory.push("czerwony");  
kolory.push("zółty");  
  
// możemy też bezpośrednio dodawać elementy  
tablicy  
kolory[3] = "niebieski";  
// wyciągamy teraz kolory  
for (var i=0; i < kolory.length; i++){  
    alert("kolor: " + kolory[i]);  
}
```


Tablice

W JavaScript tablice najlepiej jest definiować od razu jako utworzenie nowego obiektu typu Array:

```
// pusta tablica - niezadeklarowana
var tablica = new Array();

// tablica 5 elementów bez zadeklar. wartości
var tablica = new Array(5);

// tablica 3 elementów zadeklarowanych
var tablica = new Array(2,3,5);
var tablica = new Array("a","b","c");
var tablica = new Array(2,"a",3.14);

// dodawanie kolejnych elementów tablicy
tablica.push("b",10,12.2);

// dostęp do elementu nr 2 tablicy
alert(tablica[2]); // na ekranie wartość ???
```

Tablice

W JavaScript tablice najlepiej jest definiować od razu jako utworzenie nowego obiektu typu Array:

```
// pusta tablica - niezadeklarowana  
var tablica = new Array();  
// tablica 5 elementów bez zadeklar. wartości  
var tablica = new Array(5);  
// tablica 3 elementów zadeklarowanych  
var tablica = new Array(2,3,5);  
var tablica = new Array("a","b","c");  
var tablica = new Array(2,"a",3.14);  
// dodawanie kolejnych elementów tablicy  
tablica.push("b",10,12.2);  
// dostęp do elementu nr 2 tablicy  
alert(tablica[2]); // na ekranie wartość
```



3.14

Tablice i pętla “for-in”

Bardzo często koniecznym jest przeiterowanie przez wszystkie elementy tablicy w tym celu możemy użyć pętli “for”, jednak wygodniejszą jej postacią jest pętla “for-in”

```
var tablica = new Array(3,4,5,6);  
for ( k in tablica) {  
    alert(tablica[k]);  
}
```

```
var tablica = new Array(3,4,5,6);  
var suma = 0;  
for ( k in tablica) {  
    suma += tablica[k];  
}
```

Tablice i pętla “for-in”

Bardzo często koniecznym jest przeiterowanie przez wszystkie elementy tablicy w tym celu możemy użyć pętli “for”, jednak wygodniejszą jej postacią jest pętla “for-in”

```
var tablica = new Array(3,4,5,6);  
for ( k in tablica) {  
    alert(tablica[k]);  
}
```

```
var tablica = new Array(3,4,5,6);  
var suma = 0;  
for ( k in tablica) {  
    suma += tablica[k];  
}
```



suma = 18

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakiecie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

```
// metoda .concat()
```

```
var tab1 = new Array("a", "b", "c");
```

```
var tab2 = new Array("d", "e", "f");
```

```
var tab3 = tab1.concat(tab2);
```

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakiecie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

```
// metoda .concat()
```

```
var tab1 = new Array("a", "b", "c");
```

```
var tab2 = new Array("d", "e", "f");
```

```
var tab3 = tab1.concat(tab2);
```

Metoda ta kopiuje "tab1" i dołącza do niej elementy "tab2".

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakiecie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

// metoda `.concat()`

```
var tab1 = new Array("a", "b", "c");
```

```
var tab2 = new Array("d", "e", "f");
```

```
var tab3 = tab1.concat(tab2);
```

Metoda ta kopiuje "tab1" i dołącza do niej elementy "tab2". Wynikiem jest nowa tablica "tab3" zawierająca: [a,b,c,d,e,f].

// metoda `.join()`

```
var tab1 = new Array("a", "b", "c");
```

```
var string = tab1.join('|');
```

Przydatne metody dla obiektu Array

Standardowe obiekty JavaScriptowe dostarczają w pakiecie przydatne metody pozwalające na proste operacje na danym obiekcie. Dla obiektu Array najważniejsze to:

// metoda `.concat()`

```
var tab1 = new Array("a", "b", "c");
```

```
var tab2 = new Array("d", "e", "f");
```

```
var tab3 = tab1.concat(tab2);
```

Metoda ta kopiuje "tab1" i dołącza do niej elementy "tab2". Wynikiem jest nowa tablica "tab3" zawierająca: [a,b,c,d,e,f,].

// metoda `.join()`

```
var tab1 = new Array("a", "b", "c");
```

```
var string = tab1.join('|');
```

Tworzy łańcuch tekstowy z elementów tablicy z separatorem podanym jak argument metody.

Przydatne metody dla obiektu Array

```
// metoda .pop()
```

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

Przydatne metody dla obiektu Array

```
// metoda .pop()
```

```
var tabl = new Array("a", "b", "c");  
var ostatni = tabl.pop();
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

Przydatne metody dla obiektu Array

// metoda .pop()

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

// metoda .push()

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Przydatne metody dla obiektu Array

// metoda .pop()

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

// metoda .push()

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Dodaje na końcu tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

Przydatne metody dla obiektu Array

// metoda .pop()

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

// metoda .push()

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Dodaje na końcu tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

// metoda .reverse()

```
var tab1 = new Array("a", "b", "c");  
var tab3 = tab1.reverse();
```

Przydatne metody dla obiektu Array

// metoda .pop()

```
var tab1 = new Array("a", "b", "c");  
var ostatni = tab1.pop();
```

Usuwa ostatni element tablicy i jednocześnie zwraca jego wartość.

// metoda .push()

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Dodaje na końcu tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

// metoda .reverse()

```
var tab1 = new Array("a", "b", "c");  
var tab3 = tab1.reverse();
```

Odwraca kolejność elementów tablicy.

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

```
var tabl = new Array("a", "b", "c");  
var pierwszy = tabl.shift();
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

```
// metoda .unshift()
```

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Przydatne metody dla obiektu Array

```
// metoda .shift()
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

```
// metoda .unshift()
```

Dodaje na początku tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Przydatne metody dla obiektu Array

// metoda `.shift()`

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

// metoda `.unshift()`

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Dodaje na początku tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

// własność `.length`

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.length;
```

Przydatne metody dla obiektu Array

// metoda `.shift()`

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.shift();
```

Usuwa pierwszy element tablicy i jednocześnie zwraca jego wartość.

// metoda `.unshift()`

```
var tab1 = new Array("a", "b", "c");  
var tab2 = new Array("d", "e", "f");  
var tab3 = tab1.push(tab2);
```

Dodaje na początku tablicy elementy zapisane jako argument metody. Zwraca długość tablicy po modyfikacji.

// własność `.length`

```
var tab1 = new Array("a", "b", "c");  
var pierwszy = tab1.length;
```

To jest własność która zwraca długość tablicy.

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste

Obiekty

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste

Są to liczby, łańcuchy oraz wartości logiczne, posiadające metody, ale są niezmienne.

VS.

Obiekty

Są to asocjacyjne kolekcje klucz-wartość, które można dowolnie modyfikować.

Obiekty

Poznaliśmy już jeden typ obiektów w języku JavaScript jakim są tablice. W języku JavaScript wszystkie wartości poza prostymi typami liczbowymi, łańcuchowymi lub logicznymi są obiektami.

Typy proste

Są to liczby, łańcuchy oraz wartości logiczne, posiadające metody, ale są **niezmienne**.

VS.

Obiekty

Są to asocjacyjne kolekcje klucz-wartość, które można dowolnie **modyfikować**.

W języku JavaScript:

- tablice są obiektami,
 - funkcje są obiektami,
 - wyrażenia regularne są obiektami,
 - obiekty są obiektami.
-

Tworzenie obiektów

Jedną z podstawowych umiejętności w JavaScript jest tworzenie własnych obiektów reprezentujących logicznie powiązane kolekcje danych.

Obiekt a w zasadzie “literał obiektowy” jest zapisywany za pomocą nawiasów klamrowych który zawiera zero lub więcej par klucz (nazwa)-wartość:

```
var obiekt1 = {};
```

```
var obiekt2 = {
```

```
    imie: "Marcin",
```

```
    nazwisko: "Zielinski"
```

```
};
```

klucz

wartość

Uwaga:

Nazwa własności może być dowolnym słowem, jednak kiedy jest to np. słowo zastrzeżone dla składni języka JavaScript powinno być pisane w cudzysłowie.

Tworzenie obiektów

Inny przykład:

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```

Tworzenie obiektów

Różnica w pisaniu nazw własności (kluczy) :

```
var flight = {  
    carrier: "LOT",  
    number: 3911,  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        "departure-time": "22:45",  
        "arrival-time": "23:35"  
    }  
};
```

Uwaga:

W przypadku wybierania nazw własności (kluczy) w których separatorem jest puza (-) obowiązkowo należy pisać te nazwy w cudzysłowach.

Dostęp do pól obiektu

Mamy dwa obiekty:

```
var flight = {  
  carrier: "LOT",  
  number: 3911,  
  origin: "WAW",  
  destination: "KRK",  
  schedule: {  
    "departure-time": "22:45",  
    "arrival-time": "23:35"  
  }  
};
```

```
var flight = {  
  carrier: "LOT",  
  number: 3911,  
  origin: "WAW",  
  destination: "KRK",  
  schedule: {  
    departure_time: "22:45",  
    arrival_time: "23:35"  
  }  
};
```

Dostęp do pól obiektu

Mamy dwa obiekty:

```
var flight = {  
  carrier: "LOT",  
  number: 3911,  
  origin: "WAW",  
  destination: "KRK",  
  schedule: {  
    "departure-time": "22:45",  
    "arrival-time": "23:35"  
  }  
};
```

```
var flight = {  
  carrier: "LOT",  
  number: 3911,  
  origin: "WAW",  
  destination: "KRK",  
  schedule: {  
    departure_time: "22:45",  
    arrival_time: "23:35"  
  }  
};
```

Aby pobrać dane z konkretnego pola w obiekcie najczęściej stosuje się notację z “kropką” która jest separatorem np.:

```
var numer_lotu = flight.number;  
alert(flight.origin);  
alert(flight.schedule["arrival-time"]);
```

Dostęp do pól obiektu

Mamy dwa obiekty:

```
var flight = {  
  carrier: "LOT",  
  number: 3911,  
  origin: "WAW",  
  destination: "KRK",  
  schedule: {  
    "departure-time": "22:45",  
    "arrival-time": "23:35"  
  }  
};
```

```
var flight = {  
  carrier: "LOT",  
  number: 3911,  
  origin: "WAW",  
  destination: "KRK",  
  schedule: {  
    departure_time: "22:45",  
    arrival_time: "23:35"  
  }  
};
```

Aby pobrać dane z konkretnego pola w obiekcie najczęściej stosuje się notację z “kropką” która jest separatorem np.:

```
var numer_lotu = flight.number;  
alert(flight.origin);  
alert(flight.schedule["arrival-time"]);  
alert(flight.schedule.arrival_time); // dla przykładu po  
// prawej może być tak
```

Modyfikacja własności obiektu

Obiekt można zmodyfikować przez przypisanie. W przypadku gdy dana własność istnieje zostanie ona nadpisana nową wartością, w przypadku kiedy jeszcze nie istnieje zostanie dodana do obiektu:

```
flight.number = 3913; // zmiana istniejącej wartości  
flight.equipment = "388";  
  
console.log(flight); // polecenie dla konsoli np. w  
                      // Firebugu
```

Modyfikacja własności obiektu

Obiekt można zmodyfikować przez przypisanie. W przypadku gdy dana własność istnieje zostanie ona nadpisana nową wartością, w przypadku kiedy jeszcze nie istnieje zostanie dodana do obiektu:

```
flight.number = 3913; // zmiana istniejącej wartości  
flight.equipment = "388";
```

```
console.log(flight); // polecenie dla konsoli np. w  
                     // Firebugu
```

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment: "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```


Usuwanie własności obiektu

Ostatnią istotną operacją jaką można wykonać na obiekcie jest usuwanie własności. Do tego celu służy operator “delete”:

```
delete flight.number; // usuwamy własność number  
delete flight.carrier; // usuwamy nazwę przewoźnika
```

Usuwanie własności obiektu

Ostatnią istotną operacją jaką można wykonać na obiekcie jest usuwanie własności. Do tego celu służy operator “delete”:

```
delete flight.number; // usuwamy własność number  
delete flight.carrier; // usuwamy nazwę przewoźnika
```

Postać obiektu po usunięciu dwóch własności:

```
var flight = {  
    equipment = "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    }  
};
```

Zmienne globalne

W JavaScript możemy przechowywać całą zawartość aplikacji w zmiennych globalnych. Gdy chcemy zastosować zmienne globalne możemy:

```
var globalObj = {}; // tworzymy nowy pusty obiekt który
                    //będzie kontenerem dla naszej aplikacji
globalObj.pax = {
    first_name: "Marcin",
    last_name: "Zielinski"
};
globalObj.flight = {
    carrier: "LOT",
    number: 3913,
    equipment: "388",
    origin: "WAW",
    destination: "KRK",
    schedule:{
        departure_time: "22:45",
        arrival_time: "23:35"
    }
};
```

Zmienne globalne

W JavaScript możemy przechowywać całą zawartość aplikacji w zmiennych globalnych. Gdy chcemy zastosować zmienne globalne możemy:

```
var globalObj = {}; // tworzymy nowy pusty obiekt który
                    //będzie kontenerem dla naszej aplikacji
globalObj.pax = {
    first_name: "Marcin",
    last_name: "Zielinski"
};
globalObj.flight = {
    carrier: "LOT",
    number: 3913,
    equipment: "388",
    origin: "WAW",
    destination: "KRK",
    schedule:{
        departure_time: "22:45",
        arrival_time: "23:35"
    }
};
```

Jednak zmienne globalne zmniejszają elastyczność aplikacji i raczej się powinno ich unikać.

Zmienne globalne

```
console.log(globalObj);
```

```
globalObj = {  
  pax: {  
    first_name: "Marcin",  
    last_name: "Zielinski"  
  },  
  flight: {  
    carrier: "LOT",  
    number: 3913,  
    equipment: "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
      departure_time: "22:45",  
      arrival_time: "23:35"  
    }  
  }  
};
```

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
  carrier: "LOT",  
  number: 3913,  
  equipment = "388",  
  origin: "WAW",  
  destination: "KRK",  
  schedule: {  
    departure_time: "22:45",  
    arrival_time: "23:35"  
  }  
};
```

Nasz pierwotny
obiekt do
którego chcemy
dodać metodę.

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment = "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    },  
    showFlightNumber: function() {  
        alert(this.number);  
    }  
};
```

Stworzyliśmy
wewnątrz
obektu funkcję
nienazwaną!

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment = "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    },  
    showFlightNumber: function() {  
        alert(this.number);  
    }  
};
```

Stworzyliśmy
wewnątrz
obektu funkcję
nienazwaną!

Odwołanie do obiektu z wnętrza
samego siebie !

Funkcje w obiektach

Składnikiem obiektu może być również funkcja wykonująca dowolne operacje na elementach naszego obiektu:

```
var flight = {  
    carrier: "LOT",  
    number: 3913,  
    equipment = "388",  
    origin: "WAW",  
    destination: "KRK",  
    schedule: {  
        departure_time: "22:45",  
        arrival_time: "23:35"  
    },  
    showFlightNumber: function() {  
        alert(this.number);  
    }  
};
```

Stworzyliśmy
wewnątrz
obiektu funkcję
nienazwaną!

Odwołanie do obiektu z wnętrza
samego siebie !

```
// wywołanie funkcji będącej częścią obiektu  
flight.showFlightNumber();
```

Informacja o typie pola

Czasami zdarza się że konieczny jest znajomość typu danego pola obiektu. Można to zrobić korzystając z funkcji wybudowanej “typeof” która pobiera daną wartość i sprawdza jej typ:

Informacja o typie pola

Czasami zdarza się że konieczny jest znajomość typu danego pola obiektu. Można to zrobić korzystając z funkcji wybudowanej “typeof” która pobiera daną wartość i sprawdza jej typ:

```
typeof flight.number;    // liczba całkowita bo 3912
typeof flight.equipment; // łańcuch bo "388"

typeof flight.schedule;  // obiekt

typeof flight.showFlightNumber; // funkcja
```

Informacja o typie pola

Czasami zdarza się że konieczny jest znajomość typu danego pola obiektu. Można to zrobić korzystając z funkcji wybudowanej “typeof” która pobiera daną wartość i sprawdza jej typ:

```
typeof flight.number;    // liczba całkowita bo 3912
typeof flight.equipment; // łańcuch bo "388"

typeof flight.schedule;  // obiekt

typeof flight.showFlightNumber; // funkcja
```

Aby sprawdzić czy obiekt posiada interesującą nas własność możemy zastosować metodę “hasOwnProperty()”, która zwraca wartość logiczną true/false w zależności czy obiekt posiada interesującą nas własność:

```
flight.hasOwnProperty("number"); // true !!
flight.hasOwnProperty("showEquipment"); // false !!
```

KONIEC WYKŁADU 5
