**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**

(zajęcia 15.10.2015 r.)

1) Struktura dokumentu HTML:

Podstawowa struktura dokumentu HTML (Hiper Text Markap Language):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//PL"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title> Tytuł strony </title>
 </head>
 <body>
 Zawartość strony …
 <!-- Ewentualny komentarz w kodzie strony -->
 </body>
</html>
```

Poznane znaczniki:

Formatowanie tekstu: `<b><i><u>`

Akapit: `<p> </p>`

Znak nowej linii: `<br/>`

Osadzanie obrazka: `<img src="obrazek.png"/>`

Tu wzmianka o atrybutach znaczników w ogólności:

`<znacznik atrybut = " wartość " >`

2) Kontynuujemy podstawowe znaczniki:

Tytuły: `<h1> </h1> <h2> </h2> … <hn> </hn>  (1...6)`

Wyśrodkowanie tekstu/obrazka: `<center> </center>`

Linia pozioma: `<hr/>`

Listy wyliczeniowe (nienumerowane):

```
<ul>
     <li> Pozycja 1 </li>
     <li> Pozycja 2 </li>
     <li> Pozycja 3 </li>
</ul>
```

Przy listach wyliczeniowych nienumerowanych można zdecydować jaki typ symbolu ma być widoczny przy danej pozycji listy np.

`<ul type="disc">` - pełne koło (domyslny),

`<ul type="circle">` - puste koło,

`<ul type="square">` - kwadrat.

Listy numerowane:

```
<ol>
     <li> Pozycja 1 </li>
     <li> Pozycja 2 </li>
     <li> Pozycja 3 </li>
</ol>
```

3) Kodowanie na stronach WWW

W celu poprawnego wyświetlania treści dokumentu hipertekstowego należy w nagłówku strony zadeklarować kodowanie czyli. według jakiego schematu komputer ma zapisywać znaki w pamięci. W polsce obowiązującym schematem kodowania jest ten zgodny ze standardem ISO-8859-2. Do prawidłowej deklaracji kodowania używamy tzw. meta-tagów (meta-znaczników): `<meta>`. W przypadku kodowania znacznik ten ma postać:

```
<meta http-equiv="Content-type" content="text/html; charset=iso-8859-2"/>
```

Natomiast obecnie bardzo dobrze sprawdzającym się standardem kodowania jest UTF-8 oparty o system UNICODE. Aby go zadeklarować w dokumencie hipertekstowym należy wpisać:

```
<meta http-equiv="Content-type" content="text/html; charset=utf-8"/>
```

Aby znaki były poprawnie wyświetlane w dokumencie hipertekstowym używany do jego edycji program musi obsługiwać dany system kodowania!!!

Dygresja o meta-tagach:
Znaczniki te pozwalaja na umieszczenie w nagłowku stronu kilku dodatkowych informacji np. o zawartości, słowach kluczowych, autorze strony. Informacje te mogą być przydatne np. dla wyszukiwarek. Chociaż obecnie wyszukiwarki zamiast przeszukiwać zawartośc meta-tagów skupiają się na kontencie i prawidłowym osadzeniu np. znaczników <h1>.
Dla porządku podajemy jak powinny wyglądać trzy najważniejsze meta-znaczniki:

```
<meta http-equiv="Content-type" content="text/html; charset=utf-8"/>
<meta name="description" content="Ta strona zawiera informacje o HTML" />
<meta name="keywords" content="HTML, XHTML, JavaScript, PHP, MySQL" />
<meta name="author" content="Marcin Zieliński">
```

- *Ćwiczenie z dodaniem meta tagów i kodowaniem.*

gdzie: `http-equiv`-pozwala na określenie wartości dla wybranego nagłówka HTTP (może być jeszcze np. “refresh”, lub expires).

`name` - nazwa meta tagu,

`content` - zawartość / treść znacznika.

4) Znaki specjalne / jak wstawić znak “<” “>” na stronie internetowej ??

Proszę spróbować bezpośrednio w treści <body> wpisać znak większości lub mniejszości. Przeglądarka nie zwróci błędu, a nawet wyświtli wpisany znak. Jednak jest to nieprawidłowe z punktu widzenia składni HTML i XHTML. Dla prawidłowego wyświetlania znaków zarezerwowanych dla składni jezyka HTML używa się tzw. encji. Encje reprezentują odpowiedni znak. Postać ogólna encji jest następująca:

& teskt ;

gdzie “tekst” jest odpowiedni dla reprezentacji danego znaku. Encji można używać w dowolnym tekście na stronie gdzie zachodzi potrzeba wstawienia znaku specjalnego. Przykłądy najczęściej uzywanych encji:

```
<ul>
     <li>   - twarda spacja</li>
     <li> &quot; - cudzysłów informatyczny</li>
     <li> &lt;   - znak mniejszości </li>
     <li> &gt; - znak większości </li>
     <li> &copy; - prawa autorskie </li>
     <li> &reg; - znak towarowy </li>
```

```
            <li> &amp; - ampersound </li>
            <li> &ndash; - pałza </li>
            <li> &mdash; - długa pauza </li>
        </ul>
```

- *Tu ćwiczenie z wprowadzeniem znaków specjalnych.*

5) Odnośnik (hiperłącze):

```
<a href="link.html"> Tekst odnośnika (lub obrazek) </a>
```

Ścieżki w atrybucie "href" mogą być względne lub bezwzględne. Jeśli odnosimy się do łącza zewnętrznego to podajemy ścieżkę bezwzględną np. href="http://www.onet.pl"
Natomiast jeśli odnosimy się do treści we własnej witrynie (domenie) to używamy ścieżek względnych (podobnie jak w systemie linux): np href="../doc/plik.pdf".

Odnośnik może poza usługa (protokołem) http. kierować do usługi ftp, lub poczty elektronicznej. Odpowiednie postacie to:

<a href="http://www.uj.edu.pl"> Link do strony UJ </a>
<a href="ftp://serwerFTP.pl"> Link do serwera FTP </a>
<a href="mailto:m.zielinski@uj.edu.pl"> m.zielinski@uj.edu.pl <a>

Kotwice wewnątrz danej strony:

Kotwice służą do przechodzenia pomiędzy treściemi w ramch tej samej strony lub tej samej domeny. W celu użycia kotwicy należy ją "zażucic" to znaczy oznaczyć nią dany obszar tekstu do którego chcemy kierować lub jedno znadnie. Następnie w innej częscie strony umieszczamy odnośnik do tej kotwicy. Takie zastosowanie przejść pozwala nam np na zbędne przewijanie tekstu na stronach.
Użycie kotwicy jest następujące:

```
<a name="nazwakotwicy"> Obszar do zaznaczenia kotwicą </a>
```

odnoścnik do kotwicy w ramach tej samej strony tworzymy przez:

```
<a href="#nazwakotwicy"> Link do ktwicy </a>
```

Natomiast w obszarze tej samej witryny ale innej strony używamy:

```
<a href="strona1.html#nazwakotwicy"> Kotwica na innej stronnie </a>
```

- *Tu ćwiczenia z odnośnikami i kotwicami.*

6) Tebale na stronach.

Jednym z podstawowych elementów na stronach do porządkowania i przechowywania informacji sa tabele. Podstawowa deklaracja tabeli na stronie wygląda następująco:

```
<table>
    <tr>  <!-- Wiersz pierwszy -->
        <td> Komórka 1 </td> <td> komórka 2 </td>
    </tr>
    <tr>  <!-- Wiersz drugi  -->
        <td> Komórka 1 </td> <td> komórka 2 </td>
    </tr>
</table>
```

W standardowej wersji jest tabele ma atrybut border ustawiony na 0. W celu jego zmiany

należy ręczenie to zadeklarować: `<table border=”1”>`
Odstępy w komórkach można osiągnąć przez: `<table cellpadding="10px">`
Odstępny pomiędzy komórkami: `<table cellspacing="10px">`
Można również ustawić nagłówki w kolumnach przez dodanie dodatkowego wiersza na początku tabeli:

```
<tr>
<th>nagłówek kolumny 1</th>     <th>Nagłówek kolumny 2</th>
</tr>
```

- *Ćwiczymy zrobienie listy/tabeli/osadzenia obrazka/hiperłącza/kotwicy.*

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**

(zajęcia 22.10.2015 r. i 29.10.2015 r.)

## 0) Podstawowa struktura dokumentu hipertekstowego HTML

```
<!DOCTYPE html>
<html lang="pl">
<head>
<title>Witamy w HTML 5</title>
<meta charset="utf-8" />
<script src="js_file.js"></script>
<link rel="stylesheet" href="css_file.css" />
</head>
<body>
<!-- Tutaj będą m.in. nowe elementy HTML 5 -->
<h1>Witamy w HTML 5</h1>
</body>
</html>
```

Podstawowe nowe znaczniku ułatwiajace tworzenie struktury / konspektu strony:
<header> – do wyodrębnienia nagłowka strony
<nav>, – do wyodrębnienia nawigacji strony
<article> – do wyodrębnienia części strony - np post na blogu
<section>, – też do grupowania treści
<footer>, – do tworzenia stopki
<aside> – do tworzenia treści dodatkowej (np na prawo storeny )

```
<!DOCTYPE html>
<html lang="pl">
<head>
<title>Witamy w HTML 5</title>
<meta charset="utf-8" />
<style rel="stylesheet">
      article, aside, nav {display: block;}
      article, aside {float: left;}
      article {width: 500px;}
      nav {width: 200px;}
</style>
</head>
<body>
<!-- Tutaj będą m.in. nowe elementy HTML 5 -->
<article>
<header>
  <img alt="HTML 5" src="logo-small.gif" width="200" height="40" />
  <h1>Witamy w HTML 5</h1>
</header>
<p style="font-family: Arial;">
<span class="html">HTML 5</span> oferuje wiele nowych możliwości. Jednak
minie trochę czasu, zanim możliwości te będą w pełni dostępne.
```

```
</p>
<footer>
<small>&copy; Projekt HTML 5</small>
</footer>
</article>
<aside>
  <h2>Polecane linki</h2>
  <nav>
    <ul>
       <li><a href="#">Strona główna</a></li>
       <li><a href="#">O witrynie</a></li>
       <li><a href="#">Projekt</a></li>
       <li><a href="#">Wiadomości</a></li>
       <li><a href="#">Kontakt</a></li>
     </ul>
  </nav>
</aside>
</body>
</html>
```

**1) Kaskadowe arkusze stylów - CSS:**

Pozwalają na oddzielenie warstwy formatu od warstwy danych (treści).
Stylizacja strony (witryny) odbywa się w osobnym pliku.

W arkuszach stylów stosujemy formułe:

```
selektor1, selektor2, ... { cecha1: wartość;
                            cecha2: wartość;
                            cecha3: wartość;
                           …;
                         }
```

Selektor - jest to nazwa identyfikująca znacznik dla którego przeglądarka ma przypisać styl.
Cecha - Włsność jaką chcemy określić dla danego selektora.
Wartość - ...

**2) Deklaracja styli CSS - trzy metody:**

a) Syle wbudowane (inline-styles) w danym znaczniku:

```
<h1 style="font-family: Arial;
           text-align: center;
           color: green;
           background: yellow;">
     To jeszcze jeden nagłówek 1-go stopnia ze stylem osadzonym.
</h1>
```

lub `<body style="background: blue;">` / kolorujemy tło sekcji body.

b) Style osadzone (embedded-styles) wewnątrz sekcji head:

```
<style rel="stylesheet" type="text/css">
       h1 {text-align: center;
           color: yellow;
           background: green;
           padding: 4px;}
       p.uwaga_12{font-size:16px;
           font-family:Arial;
           color:#804040;
           font-weight:bold;}
</style>
```

c) Deklaracja w pliku zewnętrznym - linked-style (najczęściej stososwana i najlepsza)

```
<head>
  <link rel="stylesheet" type="text/css" href="style1.css" />
<head>
```

**3) Dziedziczenie:**

Najistotniejszy z punktu widzenia dziedziczenia i stosowalności danej cechy-własności jest styl zadeklarowany "inline", w drugiej kolejności "osadzony, a na końcu "dołączony".

Dodatkowo można deklarować selektory tak aby zastosowanie danej własności miało miejsce tylko wtedy kiedy znajduje się wewnątrz innego selektora:

```
div p { text-align: center;}      // to dwie rożne deklaracje
div, p { text-align: center;}     //
```

**4) Klasy i identyfikatory selektorów w CSS:**

a) Klasy stosujemy aby rozróżnić / nadać / tym samym selektorom wystepującym w danym dokumencie hipertekstowym różne cechy np. w witrynie częstowo występują znaczniki <h1> ale dla zabawy chcemy aby w zależności od ważności danego paragrafu były pisane różnymi kolorami. CO WTEDY?? ----> Wtedy stosujemy klasy selektorów!!
Ogólna deklaracja klasy selektora wygląda następująco:

```
selektor.klasa { cecha: wartość; } // klasy stowarzyszone z selektorem
```

aby odwołać się do konkretnej kalsy w tekście to należy wybrać klase slektora przy danym znaczniku:

```
<selektor class="klasa"> … </selektor>
```

A więc w naszym przykładzie będzie:

```
<style type="text/css">
      h1 { font-family: Verdana; }
      h1.wazny { color: blue; }     // monżna deklarować HEXem
      h1.bwazny { color: red; }     // #FFFFFF - biały
```

```
        h1.bbwazny {color: green;}    // #000000 - czarny
    </style>

        <h1> Tekst standardowy pisany Verdana </h1>
        <h1 class="wazny"> Tekst wazny </h1>
        <h1 class="bwazny"> Tekst bardzo wazny </h1>
        <h1 class="bbwazny"> Tekst bardzo bardzo wazny </h1>
```

W ten sposób można definiować klasy dla takich samych selektorów. Jak w tym przypadku jest z dziedziczeniem ???

Klasy nazwane nie przypisane do selektorów:

```
    .klasa { cecha: wartość; } // klasy NIE-stowarzyszone z selektorem
```

Np:

```
<style type="text/css">
   .srodek { text-align: center;}
</style>

   <p class="srodek"> tekst wysrodkowany za pomoca klasy srodek </p>
   <h1 class="srodek"> Tekst wysrodkowany </h1>
```

b) Identyfikatory to w zasadzie identyczne obiekty jak klasy pozwalające na określenie cech dla danego selektora. Rożnica jest taka że identyfikatory mogą być przypisane w danym dokumencie hiperteksotwym tylko raz do jednego selektora. Ogólna deklaracja to:

```
   selektor#identyfikator { cecha: wartość; } // identyfikator z selektorem
```

Nadanie cech dla danego selektora nastepuje przez atrybut id=””:

```
   <selektor id=””> .. </selektor>
```

Przyklad tym razem z h2:

```
<style type="text/css">
      h2 { font-family: Times; }
      h2#wazny {color: #ABCDEF; }
      h2#bwazny {color: #793732; }
      h2#bbwazny {color: #91AF72; }
  </style>

        <h2> Tesk pisny h2 </h2>
        <h2 id="wazny"> Teskt wazny </h2>
        <h2 id="bwazny"> Teskt b.wazny </h2>
        <h2 id="bbwazny"> Teskt b.b.wazny </h2>
```

**5) Stylizowanie odsyłaczy w zależnosić od danego stanu odsyłacza.**

Możliwe statusy hiperłącz:
link – podstawowy odsyłacz;

hover – odsyłacz, nad ktorym zatrzymano kursor myszy;
focus – odsyłacz z tzw. fokusem, czyli miejscem zaznaczonym przy poruszaniu się po dokumencie przy użyciu klawisza tabulacji. Naciśnięcie klawisza Enter – spowoduje taką samą reakcję przeglądarki, jakby dany odsyłacz zostal klikniety;
acive – aktywny odsyłacz (kliknięty);
visited – odsyłacz, który prowadzi do wcześniej już odwiedzonego dokumentu HTML, tj. takiego, który znajduje się w historii przeglądarki.

Możemy definiować style dla odsyłączy w zależności od przyjętego aktualnego stanu odsyłacza:

```
<style type="text/css">
   a:link {color: red;}
   a:visited {color: #9AB676;}
   a:active {color: green;}
   a:hover {color: yellow;}
  </style>
<a href="http://www.onet.pl">ONET.PL</a>
```

Można również stosować klasy:

```
<style type="text/css">
   a:link.uj {color: #00FF00;}
   a:visited.uj {color: #800000;}
   a:active.uj {color: #00FFFF;}
   a:hover.uj {color: #808000;}
</style>
<a class="uj" href="http://www.uj.edu.pl">UJ.EDU.PL</a>
```

**6) Formatowanie tekstu:**

Typy czcionek: font-family: serif, Arial, Helvetica, etc…
Rozmiary czcionek:
font-size: xx-small|x-small|small|medium|large|x-large|xx-large|larger|smaller|n_pt|n_px|n%
font-size: 50px;

| Parametry formatowania tekstu i bloków dopuszczalne w arkuszach stylów | | |
|---|---|---|
| Charakterystyka | **Znaczenie** | Opis i możliwe wartości |
| font-family | Rodzina czcionek | Kroje czcionek dostępne w przeglądarkach. Jeśli dany krój nie występuje, używana jest czcionka domyślna. |
| font-size | Wielkość czcionki | Dowolna wartość w punktach [pt], calach [in], centymetrach [cm] albo w pikselach [px]. Dodatkowo dopuszczalne są wartości: larger, smaller, xx-small, x-small, small, medium, large, x-large, xx-large. |

| | | |
|---|---|---|
| | | Dopuszczalna jest również wartość procentowa względem rozmiaru czcionki macierzystej [%]. |
| font-weight | Grubość czcionki | Dopuszczalne wartości: normal, bold, bolder, lighter. |
| font-style | Styl czcionki | Dopuszczalne wartości: normal, italic (kursywa), oblique (skośna). |
| font-variant | Odmiana czcionki | Dopuszczalne wartości: normal, small-caps. |
| color | Kolor czcionki. | Jeden z predefiniowanych kolorów lub kod heksadecymalny RGB. |
| background-attachment | Połączenie z tłem | Obraz przewija się razem z zawartością: scroll; obraz tła pozostaje nieruchomy: fixed. |
| background-color | Kolor tła | Przezroczyste tło: tranparent; jeden z predefiniowanych kolorów lub kod heksadecymalny RGB. |
| background-image | Obrazek tła | Żaden obrazek: none; adres URL obrazka. |
| background-repeat | Powtarzanie tła | Powtarzanie tła tylko w kierunku poziomym: repeat-x; powtarzanie tła tylko w kierunku pionowym: repeat-y; powtarzanie tła w obu kierunkack: repeat; nie powtarza tła: no-repeat. |
| border-color | Kolor krawędzi | Jeden z predefiniowanych kolorów lub kod heksadecymalny RGB. |
| border-style | Styl krawędzi | zadna krawędź: none; linia przerywana: dashed; linia kropkowana: dotted; linia ciągła: solid; linia podwójna: double; wyżłobione: groove; wypukłość: ridge; wkładka: inset; wypustka: outset. |
| border-bottom-width | Szerokość dolnej krawędzi | Cienka: thin; średnia: medium; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]. |

| | | |
|---|---|---|
| border-left-width | Szerokość lewej krawędzi | Cienka: thin; średnia: medium; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]. |
| border-right-width | Szerokość prawej krawędzi | Cienka: thin; średnia: medium; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]. |
| bottom-top-width | Szerokość górnej krawędzi | Cienka: thin; średnia: medium; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]. |
| float | Przesunięcie | Przesuwa ustawianą zawartość do w lewo: left, w prawo: right albo nie przesuwa: none. |
| padding-bottom | Odstęp od dolnej krawędzi | Dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent szerokości elementu nadrzędnego [%]. |
| padding-left | Odstęp od lewej krawędzi | Dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent szerokości elementu nadrzędnego [%]. |
| padding-right | Odstęp od prawej krawędzi | Dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent szerokości elementu nadrzędnego [%]. |
| padding-top | Odstęp od górnej krawędzi | Dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent szerokości elementu nadrzędnego [%]. |
| text-align | Wyrównanie tekstu | Lewostronny: left; wyśrodkowany: center; prawostronny: right; wyjustowany: justify. |
| text-decoration | Efekty dodatkowe | Żadne: none; podkreślenie: underline; linia nad tekstem: overline; przekreślenie: line-through; miganie: blink. |
| text-indent | Wcięcie | Dowolna liczba punktów [pt], cali |

| | | |
|---|---|---|
| | tekstu | [in], centymetrów [cm] albo pikseli [px]; procent szerokości elementu nadrzędnego [%]. |
| text-transform | Przekształcenie tekstu | Kapitaliki: capitalize; duże litery: uppercase; małe litery: lowercase; żadne: none. |
| line-height | Wysokość wiersza | Normalna: normal; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent wartości rozmiarów czcionki [%]. |
| letter-spacing | Rozstaw liter | Normalny: normal; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]. |
| word-spacing | Odstępy miedzy wyrazami | Normalny: normal; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]. |
| margin-left | Lewy margines | Automatyczny: auto; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent wartości z szerokości elementu nadrzędnego [%]. |
| margin-right | Prawy margines | Automatyczny: auto; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent wartości z szerokości elementu nadrzędnego [%]. |
| margin-top | Górny margines | Automatyczny: auto; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent wartości z wysokości elementu nadrzędnego [%]. |
| margin-bottom | Dolny margines | Automatyczny: auto; dowolna liczba punktów [pt], cali [in], centymetrów [cm] albo pikseli [px]; procent wartości z wysokości elementu nadrzędnego [%]. |
| vertical-align | Wyrównanie w pionie | Do linii bazowej: baseline; do góry: top; do góry tekstu: text-top; do środka: middle; do dołu tekstu: text-bottom; procentowa wartość bieżącej szerokości wiersza [%]; do wysokości indeksu dolnego: |

| | | |
|---|---|---|
| | | sub; do wysokości indeksu górnego: super. |

**7) Tworzenie menu za pomocą listy uporządkowanej.**

Aby stworzyć menu moża do tego celu wykorzystać liste uporządkowaną <ul> wraz z zastosowaniem odnośników - hiperłączy <a>. Najprostsza konstrukcja:

```
<ul id="menu">
  <li><a href="http://www.onet.pl"> Hiperłącze do Onet.pl </a></li>
  <li><a href="http://www.wp.pl"> Hiperłącze do Wp.pl </a></li>
  <li><a href="http://www.gazeta.pl"> Hiperłącze do Gazeta.pl </a></li>
 </ul>
```

W ten prosty sposób powstaje najprostsze menu z odnośnikami do innych stron. Aby lepiej wyglądało można go ostylizować za pomocą CSS. (cały kod w załączeniu oraz koza/~marcin/TWWW/index1.html Aby ostylizowac musimy ostylizować element <ul>, <li> oraz <a> które składają się na najprostsze menu.

Tu o marginesach na stronie - model pudełkowy:

- margines zewnętrzny: margin u b l r;
  margin-left: 10px;
- obramowanie: ramka: border: width style color;
  border-style: solid;
  border-color: #XXXXXX;
  border-width: 10px;
  border-top-style: dashed;
  border-bottom-color: #XXXXXX;
  border-left-width: 2px;
- margines wewnętrzny: padding: u b l r;
  padding-right: 2px;

Ważniejsze cechy:

list-style: none; -> zmana puntorów na liście <li>

display: block; -> jak przeglądarka ma wyświetlac elementu: blok teksu, linia tekstu, element listy. Możliwe wartości: block, inline, none (w tym kontekście visibility:hidden;).

text-decoration: none; - brak podkreślenia odnośnika.

Podobnie tworzymy menu poziome: z tą różnicą że elementom <li> musimy nadać własność float:left; (right, none).
To powoduje że elementy z tą cechą będą dryfować w zdefiniowaną stronę.

**8) Model pudełkowy / przyklad strony z nagłówkiem, sidebarem i stopką.**

Mamy już menu teraz można zrobic szkielet strony w oparciu o .
Różnica pomiędzy  a  ??

Zwykle do budowy stron używamy elementów blokowych .

```
<div style="clear:both;"></div>
```

Aby uniknąć tego efektu pływania z opcji float należy jakiemuś elementowi **za** elementem z float nadać właściwość clear — powoduje ona ponowne „złączenie”float z biegiem dokumentu.

**left**

dany element będzie zsunięty poniżej wszystkich z float:left, ale pozwoli na opływanie go z prawej.

**right**

element będzie zsunięty poniżej wszystkich z float:right, ale pozwoli na opływanie go z lewej.

**both**

element będzie zsunięty poniżej wszystkich poprzedzających go floatów

**none**

Domyślna wartość oznaczająca brak wpływu na floaty (wyłącza działanie clear).

### 9) Jeszcze na temat selektorów i reguł w CSS:

a) selektr uniwersalny: * {cecha: wartość;}

```
* {
    color:red;        }
```

Stosuje się często do zerowania np paddingów i marginesów na początku dokumentu:

```
* { margin:0 ;
    padding: 0;
  }
```

b) selektor potomka: zadziała tylko jeśli  jest potomkiem . Potomek znaczy znajduje się w tym selektorze na dowolnym poziomie zagnieżdzenia (potomek w dowolnym pokolenu).

```
div span {
          color:red;
         }

<div>
     <span> tttt </span>
</div>

<div>
     <p>
          <span> ttt </span>
     </p>
</div>

<p> <span> tekst 3 </span></p>
```

c) selektor dziecka: zadziałą tylko dla pierwszego zagnuieżdzania:

```
 div > span {
             color: green;
  }
```

Może być zbudowany z wiekszej ilości selektorów: div > p > span {}

d) Selektor brata: obejmuje swoim działaniem jeden selektor znajdujący się bezpośrednio za danym selektorem:

```
div + span { color: blue; }
<div> </div> <span> tekst </span>
```

Można również składać selektroy braci:

```
div + p + span { color: red;}
<div> </div> <p> </p> <span> xxxx </span>
```

e) selektor braci: zadziała dla wszytkich elementów danego typu za danym selektorem:

```
div ~ span { color: red;}


 <div> </div>
<span> czerwony </span>
<span> czerwony</span>
<span>czerwony</span>

<div> <span> czarny </span></div>

<hr/>

<span> czerwony </span>

<div> <p> <span> czarny </span></p> </div>
```

---

Selektory atrybutów:

a) selektor atrybutu:

```
selektor[atrybut] {cecha: wartosc;}

div[title] {color:red;}

<div title="www"> czerwony </div>
<hr/>
<p title="www"> czarnyy </p>
```

lub

```
[atrybut] {cecha: wartosc;}

 [title] {color:red;}

<div title="www"> czerwony </div>
<hr/>
<p title="www"> czerwony </p>
```

b) selektor atrybutu z wartością:

```
      selektor[atrybut=”wartość”] {cecha: wartosc;}
             [atrybut=”wartość”] {cecha: wartosc;}
        [title="www"] { color: red;
                       }

            <div title="www"> tttttt </div>
            <hr/>
            <p title="qqq"> kkk </p>
```

c) selektor dla atrybutu zawierającego dany wyraz:

```
selektor[atrybut~=”wartość”] {cecha: wartosc;}
            [atrybut~=”wartość”] {cecha: wartosc;}

    [title~="www"] { color: red;
        }

    <div title="www"> tttttt </div>
    <hr/>
    <p title="www qqq"> kkk </p>
```

d) selektor atrybutu z wyrazem z myslnikiem (łącznikiem):

```
      selektor[atrybut|=”wartość”] {cecha: wartosc;}
             [atrybut|=”wartość”] {cecha: wartosc;}

[title|="www"] { color: red;
        }

<div title="www-kkkkkkk"> tttttt </div>
    <hr/>
    <p title="www-qqq"> kkk </p>
    <hr/>
    <p title="www-ddddddqqq"> kkk </p>
```

e) selektor atrybutu zawierający dany ciąg znaków:

```
selektor[atrybut*=”wartość”] {cecha: wartosc;}
            [atrybut*=”wartość”] {cecha: wartosc;}


      [title*="www"] { color: red;
          }

  <div title="ww-kkkkkkk"> tttttt </div>
    <hr/>
    <p title="www-qqq"> kkk </p>
    <hr/>
    <p title="kkkkkk kkkww-ddddddqqq"> kkk </p>
```

f) selektor atrybutu którego wartośc zaczyna się od konretnego ciągu znaków:

```
      selektor[atrybut^=”wartość”] {cecha: wartosc;}
                      [atrybut^=”wartość”] {cecha: wartosc;}

      [title^="www"] { color: red;
         }
    <div title="www-kkkkkkk"> tttttt </div>
    <hr/>
    <p title="www-qqq"> kkk </p>
    <hr/>
    <p title="kkkkkk kkkww-ddddddqqq"> kkk </p>
```

g) selektor atrybutu którego wartość kończy się na dany ciąg znaków:

```
       selektor[atrybut$=”wartość”] {cecha: wartosc;}
                      [atrybut$=”wartość”] {cecha: wartosc;}

[title$="www"] { color: red;
        }
<div title="www-kkkkkkkwww"> tttttt </div>
    <hr/>
    <p title="www-qqq  www"> kkk </p>
    <hr/>
    <p title="kkkkkk kkkww-ddddddqqq"> kkk </p>
```

i) łączenie ze sobą warunków na kilka atrybutów dla bardziej pracyzyjnego wyboru treści której chcemy nadać odpowiednie wartości:

```
      selektor[atrybut][atrybut] {cecha: wartość;}
           [atrybut][atrybut] {cecha: wartość;}

[title$="www"][lang="pl"] { color: red;
        }
   <div title="www-kkkkkkkwww" lang="pl"> tttttt </div>
    <hr/>
    <div title="www-kkkkkkkwww" lang="en"> tttttt </div>
    <hr/>
    <p title="www-qqq  www"> kkk </p>
```

---

Selektory pseudo-elementów:

Czasem zachodzi potrzeba dostępu do określonych elementów na każdej stronie ale do których nie da się bezpośrednio odnieść: np. pierwsza litera, pierwsza linijka etc. Możaby by za kazdym razem kiedy chcemy ostylizowac pierwsza linijke np obejmować ją elementem span i stylizowac ten element ale jest to czasochłonne. Można je również wykorzystać do dodawania konkretnej treści. Ale aby rozwiązać ten problem można użyć tzw. pseudo-elementów:

a) zacznijmy od pierwszej lini:  first-line

```
        selektor:first-line { color: red;
```

```
        }
:first-line { color: red;
        }
```

Można z jednym dwukropkiem też zadziałą ale dla odróżnienie od pseudo-klas lepiej pisać z dwoma dwukropkami.

b) podobnie można zastosować: first-letter

```
selektor:first-letter{ color: red;
        }
:first-letter { color: red;
        }
```

c) pseudo element: before - dodaje tekst do każdego elementu określonym slektorem:

```
div:before {content:" --Dodaj ten tekst --";
            color: red;
    }
```

d) podobny efekt możemy osiągnąć za pomocą: after

```
div:after {content:" --Dodaj ten tekst --";
            color: red;
    }
```

e) zaznaczenie (nie działa we wszystkich przeglądarkach np. dopiero w IE9):

```
 div::selection {
    color:red;
  }
```

wtedy zaznaczające tekst myszką powinien być on podświetlony na czerwono.

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**
(zajęcia 05.11.2015 r. i 12.11.2015 r.)

## 1) Podstawowe informacje o języku JavaScript

Aby użyć w dokumencie hipertekstowym skryptu napisanego w języku Javascript należy kod tego skryptu osadzić wewnątrz dokumentu HTML lub dołączyć go w postaci zewnętrznej biblioteki:

a.) Osadzanie skryptu wewnątrz dokumentu hipertekstowego:

```
<!DOCTYPE html>
<html>
<head>
      <title>Testowanie JS</title>
<script type="text/javascript">
<!--
// tu bedzie pierwszy scrypt napisany w JS
alert("Hello World");
document.write("<h3>To jest tekst wygenerowany w js przed zdarzeniem
onLoad();</h3><hr/>");
//-->
</script>
<noscript>
<center style="color:red;">!!!! Twoja przeglądarka nie obsługuje lub ma
wyłączony JS !!!!</center>
</noscript>
</head>
<body>
<h2>Test JavaScriptu</h2>
Akcja wywołana przed zdarzeniem OnLoad(); w momencie ładowania strony
<hr/>
</body>
</html>
```

b.) Dołączenie skryptu w postaci zewnętrznej biblioteki w pliku “external.js”:

```
<!DOCTYPE html>
<html>
<head>
      <title>Testowanie JS</title>
<script type="text/javascript" src="external.js"></script>
<noscript>
<center style="color:red;">!!!! Twoja przeglądarka nie obsługuje lub ma
wyłączony JS !!!!</center>
</noscript>
</head>
<body>
</body>
</html>
```

**2) Zmienne i operacje artmetyczne:**

W języku JS nie ma statycznej kontroli typu zmiennej dlatego zmienne deklarujemu za pomocą słowa kluczowego "var", a typ określamy w momencie przypisania wartości zmiennej:

```
<script type="text/javascript">
<!--
var x;
x = 5;
alert(x*x);
x = "ddd";
alert(x);
x = null;
alert(x);
var imie = "Jan";
var nazwisko = "Kowalski";
alert(imie+" "+nazwisko);
document.write("<h1>"+imie+" "+nazwisko+"</h1>");
document.write("<hr/>");
document.write(typeof(imie));
document.write("<hr/>");
var a;
document.write(typeof(a));
//-->
</script>
```

- Dyrektywa "`document.write()`" pozwala na pisanie dowolnego tekstu do części body dokumentu hipertekstowego.
- metoda "`alert()`" pozwala na wypisanie informacji na ekran w postaci "wyskakującego okna komunikatu".
- metoda pozwala "`typeof()`" na sprawdzenie typu zmiennej.

Parsowanie zmiennych pozwala np. na zmiane typu zmiennej z String na Integer, lub z String na Float, służą do tego dwie metody "`parseInt()`" oraz "`parseFloat()`":

```
<script type="text/javascript">
<!--
var x = "1";
var y = "4";
x = parseInt(x);
y = parseInt(y);
document.write("ParseInt "+(x+y));
document.write("<hr/>");
x = "1.5";
y = "4.2";
x = parseFloat(x);
y = parseFloat(y);
document.write("ParseFloat "+(x+y));
document.write("<hr/>");
```

```
//-->
</script>
```

### 3) Tablice numerowane indeksem naturalnym:

```
var tab1 = new Array(); // tablica bez deklaracji
var tab2 = new Array(5); // 5 elementow tablica bez deklaracji wartosci
elementow
var tab3 = new Array(1,2,3,4,5);

// tablice nie maja ograniczenie na typ a wiec mozna:
var tab4 = new Array(2001, "Marcin", "Zielinski", 30, 5.0);
```

### 4) Pętle wyliczeniowe:

```
// petla for
for(n=0; n<5; n++){
 document.write(n);
 document.write("<hr/>");
}

//petla forin - wybieranie z tablicy
var tablica;
tablica = new Array("a","b","c","d","e");
for (n in tablica){
 document.write(n);
 document.write(tablica[n]+"<br/>");
}
```

### 5) Tablice asocjacyjne:

```
// Tablice asocjacyjne przypisanie wartości do nieistniejącej właściwości
obiektu
// nie skończy się błędem ale zostanie utworzona nowa właściwość obiektu i
przypisana
// zostanie do niej wartość.
var tab5 = new Object();
tab5["Ala"] = "kot";
tab5["Pi"] = 3.1415;
document.write(tab5["Ala"] + "<br>" + tab5["Pi"]);
document.write("<hr/>");

// Tablice asocjacyjne - kolejny przyklad
var tablica = new Object();
tablica["Ala"] = "kot";
tablica["Pi"] = 3.1415;
for (var klucz in tablica)
    document.write(klucz + ": " + tablica[klucz] + "<br>");
```

**6) Funkcje i metody:**

```
// definiowanie funkcji
function Wypisz(imie){
 document.write("Witaj "+imie);
}
Wypisz("Jan");

// funkcja która coś zwraca
function Dodaj(a,b){
 return a+b;
}
document.write(Dodaj(5,7));

// konstruktor
function Wspolrzedne(x,y){
      this.x = x;
      this.y = y;
}
// tworzenie obiektu klasy -> operator new
var punkt = new Wspolrzedne(2,5);
//odwolanie do pol obiektu przez .
document.write("X: "+punkt.x+" Y: "+punkt.y);
document.write("<hr/>");

// inny przyklad
function NaEkran(){
   document.write("Wynik: "+this.a);
}
function Potega(a){
      this.a = a*a;
    this.wypisz = NaEkran;
}
var zmp = new Potega(5);
zmp.wypisz();
document.write("<hr/>");
```

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**
(zajęcia 19.11.2015 r.)

## 1) Zdarzenia w JavaScript

Zdarzenia są sygnałami generowanymi w chwili wykonywania ściśle określonych czynności. JavaScript dysponuje narzędziami do informowania o zdarzeniach oraz pozwala skryptom na obsługę konkretnych zdarzeń. Przykładowo, zdarzenia mogą się pojawić, gdy użytkownik postawi wskaźnik myszy na tekście hiperpołączenia, zmieni dane w formularzu albo gdy zostanie zakończone ładowanie strony.

| Nazwa Zdarzenia | Opis |
|---|---|
| abort | Zachodzi, gdy użytkownik przerywa ładowanie obrazka. |
| blur | Zachodzi, gdy miejsce wprowadzania zostaje przesunięte do innego pola formularza lub okna. |
| Click | Zachodzi, gdy użytkownik kliknie na połączeniu lub elemencie formularza. |
| Change | Zachodzi, gdy zawartość pola formularza ulegnie zmianie. |
| Error | Zachodzi, gdy podczas ładowania strony lub obrazka wystąpi błąd. |
| Focus | Zachodzi, gdy miejsce wprowadzania zostaje umieszczone w polu. |
| Load | Zachodzi, gdy ładowanie strony zostanie zakończone. |
| Mouseout | Zachodzi, gdy użytkownik przesunie wskaźnik myszy poza tekst połączenia lub aktywny obszar mapy graficznej. |
| Mouseover | Zachodzi, gdy użytkownik przesunie wskaźnik myszy poprzez tekst połączenia lub aktywny obszar mapy graficznej. |
| Reset | Zachodzi, gdy użytkownik zażąda wyczyszczenia pól formularza poprzez kliknięcie na przycisku Reset. |

| | |
|---|---|
| Select | Zachodzi, gdy użytkownik wybierze jedno z pól formularza. |
| Submit | Zachodzi, gdy dane z formularza zostaja wysłane (zazwyczaj po kliknięciu przycisku Submit). |
| Unload | Zachodzi, gdy użytkownik zmienia wyświetlaną stronę. |

Kiedy coś dzieje się na stronie WWW, przeglądarka generuje zdarzenie (z ang. event). Jako skutek, powstaje obiekt zdarzenia, który to zdarzenie opisuje. Jednocześnie, badane jest, czy istnieje procedura obsługi zdarzenia – jeśli tak, jest wykonywana. Można zatem powiedzieć, że procedury obsługi zdarzeń są to pisane przez programistę skrypty, będące dodatkowymi argumentami określonych etykiet HTML. Te, tworzone przez programistę procedury obsługi wykonywane są w chwili wystąpienia odpowiedniego zdarzenia. Oto podstawowa składnia JavaScriptu, używana do definiowania procedur obsługi zdarzeń.

```
<Znacznik Atrybuty ProceduraObsługi="JS_Program" >
```

Przykład:

```
<div onclick = "alert(Przecież było napisane: nie klikać!);">
```

| **Nazwa zdarzenia** | **Opis** | **Elementy HTML obsługujące zdarzenie** |
|---|---|---|
| **onabort** | Procedura zostanie uruchomiona, gdy ładowanie obrazu zostanie przerwane. | **img** |
| **onblur** | Procedura zostanie uruchomiona, gdy element utraci fokus, np. gdy miejsce wprowadzania zostaje przesunięte do innego pola formularza lub okna. | Większość elementów strony. |
| **onchange** | Procedura zostanie uruchomiona, gdy element utraci fokus i jednocześnie zmieni się zawartość tego elementu (np. pola tekstowego). | **input, select, textarea** |
| **onclick** | Procedura zostanie uruchomiona, gdy element zostanie klikniety. | Większość elementów |

| | | |
|---|---|---|
| | | strony. |
| **ondblclick** | Procedura zostanie uruchomiona, gdy element zostanie dwukrotnie klikniety. | Większość elementów strony. |
| **onerror** | Procedura zostanie uruchomiona, gdy przy ładowaniu obrazu wystąpi błąd. | **img** |
| **onfocus** | Procedura zostanie uruchomiona, gdy element otrzyma fokus. | Większość elementów strony. |
| **onkeydown** | Procedura zostanie uruchomiona, gdy zostanie naciśnięty klawisz klawiatury. | Większość elementów strony. |
| **onkeypress** | Procedura zostanie uruchomiona, gdy klawisz klawiatury zostanie naciśnięty i puszczony. | Większość elementów strony. |
| **onkeyup** | Procedura zostanie uruchomiona, gdy klawisz klawiatury zostanie puszczony. | Większość elementów strony. |
| **onload** | Procedura zostanie uruchomiona, gdy przegladarka zakończy ładowanie strony lub ramki. | **body, frameset** |
| **onmousedown** | Procedura zostanie uruchomiona, gdy klawisz myszy zostanie naciśnięty nad elementem. | Większość elementów strony. |
| **onmousemove** | Procedura zostanie uruchomiona, gdy kursor myszy jest przesuwany nad elementem. | Większość elementów strony. |
| **onmouseover** | Procedura zostanie uruchomiona, gdy kursor myszy wejdzie w obszar elementu. | Większość elementów strony. |

| | | |
|---|---|---|
| **onmouseup** | Procedura zostanie uruchomiona, gdy klawisz myszy zostanie zwolniony nad elementem. | Większość elementów strony. |
| **onreset** | Procedura zostanie uruchomiona, gdy formularz zostanie zresetowany (np. przez kliknięcie przycisku *reset*). | **form** |
| **onresize** | Procedura zostanie uruchomiona, gdy zmieni się rozmiar okna. | **body, frameset** |
| **onselect** | Procedura zostanie uruchomiona, gdy fragment tekstu zostanie zaznaczony. | **input (text, textarea)** |
| **onsubmit** | Procedura zostanie uruchomiona, gdy formularz zostanie wysłany (np. przez kliknięcie przycisku *submit*). | **form** |
| **onunload** | Procedura zostanie uruchomiona, gdy przegladarka usunie bieżący dokument. | **body, frameset** |

Przykład zdarzenia “onchange”:

```
<INPUT TYPE="text" SIZE="8" value="0" onChange="
     if(parseInt(this.value)<=5) {
          alert('Proszę podać liczbę większą od 5!');
          }
">
```

Słowo kluczowe **this** odnosi się tutaj do aktualnego obiektu. W rozważanym przykładzie tym obiektem jest bieżące pole formularza (które to pole jest jednocześnie obiektem i właściwością obiektu wyższego rzędu – formularza).

Przykład zdarzenia “onclick” na elemencie :

```
<div id="dataDiv"
       style="background-color:silver;width:200px;heigth:30px;
              text-align:center;"
       onclick="alert('Kliknąłeś w element strony!');"
  >
```

Ten sam przykład tylko z rozbudowaną funkcjonalnością:

```
<script>
function dataDivClick()
    {
      var dataDiv = document.getElementById("dataDiv");
      dataDiv.innerHTML = "Warstwa została kliknięta.";
    }
</script>
  <div id="dataDiv"
       style="background-color:silver;width:200px;heigth:30px;
              text-align:center;"
       onclick="dataDivClick();"
  >
```

Przykład utworzenia linku na zdarzeniu “onclick”:

```
<p style="cursor:pointer;"
   onclick="location.href='http://www.fais.uj.edu.pl'">
Witryna Wydziału Fizyki, Astronomii i Informatyki Stosowanej
</p>
```

### 2) Emulowanie zdarzeń:

Emulowanie zdarzeń może się okazać przydatne, gdy np. trzeba wysłać formularz bez konieczności proszenia użytkownika o kliknięcie na przycisku Submit lub gdy trzeba np. zmienić miejsce wprowadzania informacji w zależności od czynności podejmowanych przez użytkownika. I tak, po wypełnieniu formularza, użytkownik mógłby kliknąc na przycisk Zamów, co powinno wywołac odpowiedni skrypt sprawdzający, czy fomularz jest poprawnie wypełniony. Dopiero pozytywny wynik tego sprawdzenia generowałby (emulowałby) zdarzenie submit w celu ostatecznego wysłania danych z formularza do serwera.

Poniżej zestawiono listę najczęściej wykorzystywanych funkcji, dostępnych w JavaScripcie, emulujących zdarzenia:

- blur()
- click()
- focus()
- reset()
- select()
- submit()

Przykład emulowania zdarzenia “click”:

```
<form>
 <input type="checkbox" id="myCheck" onmouseover="myFunction()" onclick="
alert('kliknąłeś mnie')">
</form>
<script>
function myFunction() {
 document.getElementById("myCheck").click();
}
</script>
```

**Zadanie:**

*Prosze napisać skrypt, który będzie za pomoca zdarzenia "onload" elementu <body> losowo otwierał stonę internetową o okreslonym adresie URL (prosze zdafiniowac przynajmniej 4 takie adresy). Adresy prosze trzymac w tablicy, a realizację otwarcia strony wykonać za pomocą metody "window.open(array[], string);".*

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**
(zajęcia 26.11.2015 r.)

## 1. Biblioteka jQuery (www.jquery.com) - wprowadzenie

Biblioteka napisana w języku JavaScript o "lekkim" charakterze, obsługująca przestrzenie nazw z mechanizmem łatwej rozszerzalności umożliwiająca manipulowanie elementami struktury DOM (Document Object Model). Najnowsze wydanie stabilne:

jQuery 2.1.4 ( z 28 kwietnia 2015 r.)

Biblioteka występuje w 2 wersjach:

a. normalnej - http://code.jquery.com/jquery-2.1.4.js
b. skompresowanej - http://code.jquery.com/jquery-2.1.4.min.js

Dla zwiększenia wydajności aplikacji lepiej używać wersji (b). Opcjonalnie można używac wersji udostępnianej na zewnętrznym serwerze np.
**https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js**

Aby dodać funkcjonalności biblioteki jQuery do naszej aplikacji, musimy ją tą bibliotekę jako skrypt używając znaczników <script>:

```
<!DOCTYPE html>
<html>
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery-2.1.4.min.js"></script>
</head>
<body>
 <h1>Testujemy bibliotekę jQuery</h1>
</body>
</html>
```

Pierwszą funkcją ją możemy napisac jest wyświetlenie komunikatu:

```
<head>
<title>Pierwsza strona z jQuery</title>
<script type="text/javascript" src="jquery.2.4.1.min.js"></script>
<script type="text/javascript">
      $(document).ready(function(){
        alert('Pierwsza operacja jQuery');
      });
</script>
</head>
```

Zaprezentowana konstrukcja:

```
      $(document).ready( );
```

tworzy nowy obiekt na podstawie argumentu "document" i wykonuje na nim metodę "ready()". Jest to skrótowy zapis składni jQuery gdzie znak "$" wskazuje na przestrzeń nazw "jQuery".

Konstrukcja ta w wersji pełnej wygląda nasępująco:

Dodatkowo metoda “ready()” jest tak naprawde funkcją obsługi zdarzenia “ready”, oznaczająca załadowanie całego dokumentu hipertekstowego.  Natomiast funkcja która jest przekazywana jako argument funkcji “ready()” jest tzw. funkcją anonimową, (bez nazwy) i w tym konkretnym wypadku pełni rolę wywołania zwrotnego tzw. callbacka. Jako że większość, metod jQUery chemy wykonywać po załadowaniu dokumenty hiperteksotwego można tą konstrukcję skrócić jeszcze bardziej do:

```
$(function(){
  alert('Pierwsza operacja jQuery');
});
```

Funkcja ta realizuje proste wyświetlanie komunikaty “alert()”.

## 2. Selektory i zdarzenia w jQuery

Biblioteka jQuery wspiera operacje pozwalające na manipulowanie struktura drzewa DOM. Aby operacje te mogły być wykonywane musimy określić które elementy drzewa DOM mają zostać zmodyfikowane. W jQuery robimy to stosując te same selektory co w przypadku arkuszy styli CSS. Przykłady:

`$('div')` **wybiera** `<div> </div>`
`$('h1')` **wybiera** `<h1> </h1>`

`$('#tresc')` **wybiera** `<div id="tresc"> </div>`
`$('.tresc')` **wybiera** `<div class="tresc"> </div>`

Zatem można poruszać się po drzewie wykorzystując te same własności co w CSS. Dodatkowo obowiązują te same zależności dziedziczenia pomiędzy selektorami co w przypadku CSS.

Poza selektorami dla w jQuery możamy używać tych samych funkcji obsługi zdarzeń co w normlanym języku JavaScript umożliwiając aplikacji reagowanie na działania podejmowane przez użytkownika. Napopularniejszymi zarzeniami są: “click()”, “mouseover()”, “mouseout()”.

Przykład zastosowania selektora i zdarzenia click:

```
<script type="text/javascript">
    $(function(){
      $('#button').mouseover(function(){
          alert("Klikam w link do strony UJ");
      });
    });
</script>
```

To pokazuje jak w łatwy sposób można pisać sterowniki modyfikujące zawartość DOM.

## 3. Modyfikacja wyglądu strony za pomocą CSS

W jQuery możemy wpływać na wygląd styli CSS używając metody “css()” na dowolnym elemencie. Metoda ta przyjmuje jako dwa argumenty: cechę i wartość cechy którą chcemy nadać elementowi DOM (cecha i warość muszą być podane jako string). Przykład:

```
<script type=”text/javascript”>
      $(function(){
        $(‘span’).css(‘background’, ‘#CCC000’);
      });
</script>
```

Zatem nadaliśmy cechy pojedyńczemu elementowi. Możlwie jest również odczytywanie warości cech CSS elementów i nadawanie ich innym elementom:

```
<script type=”text/javascript”>
      $(function(){
          $(‘div#tresc’).mouseover(function(){
             $(‘p’).css(‘background’, $(this).css(‘background-color’));
          }).mouseout(function(){
             $(‘p’).css(‘background’, ‘white’);
          });
      });
</script>
```

Ważne aby odczytywana cechy była cechą JEDNOWARTOŚCIOWĄ. Pojawijące się w tym kontekście słowo kluczowe “this” wskazuje na element drzewa który wywołał funkcję obsługi zdarzenia “mousover()”.

Możemy poszczególnym elementom nadawać cechy CSS dodając ich do odpowiednich klas lub usuwając je z nich:

```
<style type=”text/css”>
    .wazny { color: red; }
</style>

<script type=”text/javascript”>
      $(function(){
        $(‘li’).mouseover(function(){
           $(this).addClass(‘wazny’);
        }).mouseout(function(){
           $(this).removeClass(‘wazny’);
       });
      });
</script>
```

Zatem możemy przygotować w CSS odpowiednie definicje reguł i w zależności od systuacji nadawać je elementom.

W powyższym przykładzie widać również ważna cechę jQuery, tzw. “łancuch wywołań” (omówiliśmy go na wykładzie)>

## 4. Ukrywanie elementów DOM

W jQuery możemy wykorzystać trzy metody do ukrywania i pokazywania elementów: “hide()”, “show()”, “toggle()”. Przykłady użycia:

```
$(function(){
  $('button#hide').click(function(){
     $('p').hide();
  });
  $('button#show).click(function(){
     $('p').show();
 });
});
```

To samo na jednym elemencie można uzyskać za pomocą metody toggle:

```
$(function(){
  $('button').click(function(){
     $('span').toggle();
  });
});
```

## 4. Modyfikacja elementów DOM

jQuery został stworzony do modyfikacji elementów DOM. Dwom najprostszymi operacjiami jakie chcemy wykonywac jest dodawanie teksu i nowych znaczników.

```
$(function(){
  $('div#tresc1').text('Nowa treść wpisana dynamicznie w
jQuery');
  $('div#tresc2').html('Nowa treść wpisana dynamicznie w
jQuery');
});
```

Różnica polega na tym że metoda “text()” sparsuje wszystkie znaki specjalne i zamieni je na enecje HTML.

```
$(function(){
  $('div#tresc1').text('<span> To jest treść</span>');
  $('div#tresc2').html('<span> To jest treść</span>');
});
```

Proszę przetestować działałanie obu metod.

## 4. Modyfikacja atrybutów elementów DOM

jQuery możemy również manipulować atrybutami konkretnych elemntów DOM, służy do tego metoda “attr()”. Przykłąd zastosowania:

```
$(function(){
  $('img').attr('src', 'obrazek.png');
});
<img src="" alt="obrazek"/>
```

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**

(zajęcia 03.12.2015 r.)

## 1. Biblioteka jQuery (dodatkowe informacje)

a. **)** Metody poruszania się po drzewie DOM i dodawanie nowych węzłów i usuwanie węzłów.
b. ) Wykorzystanie metody "**ajax()**" do asynchronicznej komunikacji klient-serwer.

Informacje do punktu a.) i b) opisane są szczegółowo w wykładzie 8.

*Zadanie 1:*

*Proszę przygotować dokument HTML w którym będzie lista nienumerowana <ul> z kilkoma pozycjami. Następnie prosze przygotować skrypt który będzie reagował na zdarzenie click() na elemencie listy i w wyniku wywołania zwrotnego usuwał daną pozycje listy.*

*Zadania 2:*

*Proszę przygotować dokument HTML z fragmentem dowolnego tekstu. W tekście powinny znajdować się słowa z dodatkowymi wyjaśnieniemi (przypisami) które pojawiją się po najechaniu na dane słowo kursorem mysze w formie tzw. okna pop-up. Funkcję tę prosze oprogramować za pomocą funkcjonalności dostępnych w bibliotece jQuery. Materiały pomocne do realizacji tego zadania dostępne są w wykładzie 8.*

## 2. Wprowadzenie do środkowiska Node.js

NodeJS jest zaawansowanym środowiskiem programistycznym do tworzenia skalowalnych i bardzo wydajnych aplikacji webowych. Środowisko NodeJS (w skrócie po prostu node) napisane jest w pełni w języku JavaScript i dostępne na wszystkie platformy systemowe: Windows, Linux, MacOS X. Aplikacje oparte o środowisko node działają w asynchronicznym systemie wejścia/wyjścia sterowane za pomoca zdarzeń i pozwala na uruchomienie kodu JavaScriptowego poza przeglądarką internetową.

System został stworzony przez Ryana Dahla w 2009 roku i od tego czasu stał sie jedną z najlepszych platform tworzenia aplikacji webowych stosowana przez wiele światowych korporacji m.in. LinkedIn, Microsoft, Yahoo, Walmart, PayPal, a w Polsce np Onet.pl i serwisy spółki Agora.

Platforma ta wykorzystuje silnik Google V8 jako interpretatora języka skryptowego JavaScript. Zastosowanie bibliotek obsługujących zapytania http oraz porty, umożliwia platformie node zachowanie się jak serwer www podobnie jak np. Apache. System posiada bardzo dobrze rozbudowany system wtyczek dostępnych w repozytorium on-line (npn) które można instalować w zależności od potrzeb.

**Instalacja:**

0) Wejść na stronę: http://nodejs.org

1) Pobrać najnowsza wersję platformy node.js, a następnie rozpakować do katalogu domowego ( > tar -zxvf nazwa_archiwum.tar.gz ) i zmienić nazwę powstałej kartoteki na "nodejs".

2) W utworzonej kartotece znajduje się kartoteka “bin” w której zlokalizowany jest główny plik wykonywalny o nazwie “node”, stanowiący interpreter kodu JS.

3) Domyślnie używamy powłoki terminala “bash”. Jeśli nie jest to domyślna powłoka należy się do niej przełączyć wpisując w terminalu polecenie:

```
> bash
```

4) Definiujemy tworzymy następujące linki symboliczne w głownej kartotece nodejs:

```
> ln -s ./bin/node node
> ln -s ./bin/npm npm
```

5) W pliku **.bashrc** (jeśli nie istnieje tworzymy taki w sowjej głównej kartotece domowej) dodajemy następującą zmienną środowiskową:

```
export PATH=$PATH:/hom/username/nodejs/bin
```

gdzie username - ocznacza nazwę użytkownika
nodejs - oznacza nazwę kartoteki instalcji nodjs

6) Następnie w głównej kartotece domowej wykonujemy polecenie:

```
> source .bashrc
```

7) Jeśli w kartotece mamy np. plik.js, zawierający skrypt w języku JS, to aby go wykonać należy uruchomić go następującym poleceniem:

```
> node plik.js
```

8) W celu sprawdzenia czy poprawnie “zainstalowaliśmy” środowisko node można wykonać polecenie:

```
> node -v
```

**Pierwszy skrypt JS dla środowiska node:**

1) Tworzymy plik “witaj.js”
2) W pliku umieszczamy jednolinijkowe polecnie skryptowe:

```
console.log("witaj swiecie!");
```

Metoda console.log()pozwala na wyświetlanie komunikatów do terminala jak np. w C++ cout.

3) Uruchamiamy wykonując polecenie:

```
> node witaj.js
```

4) Wynik działania skryptu powinien być widoczny jako komunikat w terminalu linux.

Inny przykład:

```
// tak dodajemy komentarz
var a = 7;
var b = 8;
console.log( a + b );
console.log( a - b );
console.log( a * b );
console.log( a / b );
```

W języku JavaScript zmienne definiujemy za pomocą słowa kluczowego “var”. Należy podkreślić, że JS

jest językiem zmiannych bez deklaracji typu, co powoduje że typ zmiennej jest definiowany w momencie przypisania jej wartości.

Deklaracja przez obiekty:

```
var obiekt1 = {
           a: 7
           }
var obiekt2 = {
           b: 8
           }
console.log( obiekt1.a + obiekt2.b );
var obiekt3 = {
           a: 7,
           b: 8
           }
console.log( obiekt3.a + obiekt3.b );
```

Tworzenie serwera HTTP w środowiska node:

Do stworzenia serwera http w node należy użyć wbudowanego modułu “http”. Moduły dołączamy dyrektywą require w następujacy sposób:

```
var nazwa_obiektu = require(‘nazwa_modulu’);
```

zwykle dla zachowania przerzystości “nazwa_obiektu” jest tożsama z nazwą dołączanego modułu.  W przypadku modułu http dołączenie ma postać:

```
var http = require('http');
```

Następnie używamy kodu do uruchomienia serwera http znajdującego się na stronie głownej projektu node.js:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

Ta prosta funkcja uruchamia serwer http słuchający na porcie 1337 lokalnego hosta. Wynik działania można oberzeć w przeglądarce wpisując adres: http://127.0.0.1:1337.

W przypadku naszego kodu z obiektu http, uruchamiamy metodę “createServer()”. W metodzie tej jako argument podajemy funkcję która jest pewnego rodzaju funkcją odpowiedzi (zwrotną) tzw: “callback”. Zostanie ona wykonana za każdym razem kiedy przyjdzie zapytanie do serwera pod zdefiniowany adres. W funkcji tej mamy dwie zmienne:

**req -** odpowiedzialną za przechowanie obiektu żądania http.

**res -** odpowiedzialną za przechowanie obiektu odpowiedzi http.

W naszym kodzie po zgłoszeniu żadania do serwera tworzona jest dopowiedź, najpierw przez utworzenie nagłówka a następnie wygenerowanie ostatecznej odpowiedzi metodą “end”.

Ostatnia linijka zawiera metodę “listen()”, która informuje serwer na jakim porcie ma słuchać i pod jakim adresem.

**Zadanie:**
*Proszę zmodyfikować program tak aby w ostatecznej odpowiedzi renderowana była prosta strona internetowa.*

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**

(zajęcia 17.12.2015 r.)

Dla przypomnienia uruchamianie serwera http podającego statyczną treść polegało na stworzeniu skryptu wraz z osadzonym kodem statycznej strony HTML i utworzeniem funkcji obsługującej żadania przychodzące do serwera:

```
var http = require('http');

var html = '<html>'+
           '<head>'+
           '<meta charset="UTF-8">'+
           '<title>To jest strona testowa</title>'+
           '</head>'+
           '<body>'+
           '<h1>Witaj świecie w NODE.JS</h1>'+
           '<h2>to jest strona statyczna</h2>'+
           '</body>'+
           '</html>';


var server = http.createServer(function (req, res) {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.end(html);
});

server.listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

W ramach tego kodu można tworzyć statycznie działające witryny internetowe bez konieczności instalacji serwera Apache.

Dodatkowo w celu zapewnienie sterowalności aplikacją i odpowiedzi na żadania przychopdzące od klientów w postaci adresów URL można wykorzytać informacje przekazywaną w nagłówku o wpisanym adresie i tym samym np. przygotowac podstrony serwisu:

```
var http = require('http');

http.createServer( function(req,res){
  if( req.url == '/'){
        res.writeHead(200,{'Content-Type': 'text/html'});
        res.end(' index ');
  }
  else if( req.url == '/podstrona1'){
        res.writeHead(200,{'Content-Type': 'text/html'});
        res.end('podstrona1');
  }
}).listen(1337);
```

## Instalacja środowiska Express.js

Node.JS umożliwa również tworzenie złożonych i zaawansowanych aplikacji “webowych” w architekturze MVC (Model-View-Controller). Dziś poznamy kompletne narzędzia i metody do tworzenia aplikacji w tej architekturze w środkowisku NODE.JS.

Do tworzenia aplikacji w architekturze MVC w środkowsku NODE.JS służy pakiet “Express” (strona projektu: https://github.com/visionmedia/express ). Jest to bardzo rozbudowane narzędzie ułatwiające budowanie aplikacji webowych w NODE.JS. Aby z niego skorzystać należy przeprowadzić instalację pakietu “express”  składającą się z kilku kroków:

0) Domyślnie używamy powłoki terminala “bash”. Jeśli nie jest to domyślna powłoka należy się do niej przełączyć wpisując w terminalu polecenie:

```
> bash
```

1) Za pomocą standardowej metody instalacji pakietówm należy zainstalować w środowisku node.js moduł “express”:

```
> npm install express
```

2) Dodoatkowo musimy doinstalować pakiet generatora express:

```
> npm install -g express-generator
```

Na tym etapie środowisko NODE.JS jest gotowe do tworzenie aplikacji w architekturze MVC.

## Tworzenie nowego projektu

Aby utworzyć nowy projekt należy wykonać następujące kroki:

1) Utworzyć w kartotece głównej NODE.JS nowy katalog np.:

```
> mkdir hello
```

2) Wygenerować strukturę aplikacji:

```
> express hello
```

3) Wejść do kartoteki hello:

```
> cd hello/.
```

4) Doinstalować starndardowe pakiety:

```
> npm install
```

5) Na tym etapie aplikacja jest gotowa do uruchomienia. Serwer uruchamiamy wydając polecenie:

```
> npm start
```

6) W przeglądarce, po wpisaniu adresu *“localhost:3000”* powinniśmy ujrzeć domyślną stronę projektu express.

7) Dodatkowo w konsoli w której uruchomiliśmy serwer dostajemy “logi” np.:

```
GET / 200 607ms - 270b
GET /stylesheets/style.css 200 8ms - 110b
```

które mówią o czasach wykonywania przychodzących żadań do serwera i pobieranych plikach.

Zgodnie podziałem na trójwarstwową strukturę aplikacji (Model-View-Controller) poszczególne moduły aplikacji są rozproszone w różnych kartotekach. Podstawowa struktura kartotek jest następująca:

```
-rw-r--r--  05-26 10:52 app.js              // główny plik aplikacji
drwxr-xr-x  05-26 10:52 bin                 // katalog z serwerem
```

```
drwxr-xr-x  05-26 10:52 node_modules      // dodatkowe moduły node.js
-rw-r--r--  05-26 10:52 package.json
drwxr-xr-x  05-26 10:52 public            // pliki publiczne np. css
drwxr-xr-x  05-26 10:52 routes            // kontroler
drwxr-xr-x  05-26 10:52 views             // widok
```

Kartoteka “models” nie jest standardowo tworzona.

Architektura trójwarstwowa wymusza podział na akcje i widoki. Gdzie “akcje” (znajdujące się kontrolerze) są odpowiedzialne za wykonywanie operacji i obsługę żadań przychądzących do serwera, a widoki za wyświetlanie danych przychodzących z kontrolera. Standardowo “akcje” przechowywane są w kartotece “routes”, a widoki w kartotece “views”.

Rozpocznijmy od kontrolera: w kartotece “routes” znajdują sie dwa plik jeden o nazwie “index.js” drugi “user.js”. Zajmijmy się tym pierwszym, który odpowiada za obsługę strony głównej w naszej aplikacji. Jego struktura jest bardzo prosta i zawiera tylko jedną funckję obsługującą żądania GET przychodzące do serwera.

---

***Dygresja:***

**GET ->** następuje kiedy wchodzimy na stonę (wpisujemy adres w przeglądarce) i w odpowiedzi na żadanie tego typu serwer zwraca treść do wyświetlania.

**POST ->** następuje kiedy przesyłamy dane do serwera (np. wysyłając dane wpisane do formularza) które mają zostać przetworzone.

---

```
/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});
```

W tej funkcji obsługujemy żadanie GET przychodzące od klienta (z przeglądarki), wymuszące wyświetlenie strony głównej aplikacji. W odpowiedzi na przychodząse żadanie “req” formowana jest odpowiedz “res”, która renderuje stonę “index.html” na podstawie widoku znajdującego się w kartotece “views” (ale o tym za chwile). Przenalizujmy, jakie wartości przymuje metoda “render”.
Jako pierwszy argument podajemy nazwę pliku widoku który ma zostać wyświetlony w wyniku realizacji tego żadania, a drugi parametr to obiekt z danymi przekazywanymi do widoku które mają zostać wyświetlone ostatecznie na stronie. Można to porównać do funkcji która zwraca wartość (tu w postaci złożonego obiektu) do innej funkcji.

Przejdzmy teraz do widoku. W oparciu o architekturę MVC środowisko node.js wraz z modułem “express” wymusza wprowadzenie nowej formuły tworzenie widoków. Widoki poszczególnych strone przechowywane są w plikach “jade”. Pliki te nie mają postaci standardowego kodu html, pewnego rodzaju kodu “semi-html”. Strona zbudowana jest w oparciu o głowny widok przechowywany w pliku “layout.jade” oraz plików dodatkowych obsługujących poszczególne strony będące rozszerzeniem widoku głownego. W najprostszym ujęciu plik “layout” przechowuje część “head” strony, a pliki z konkretnymi podstronami zwierają tylko sekcję “body” która jest dynamicznie podmieniana w zależności od wysłanego żadania do serwera:

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

W klasycznym pliku kodzie html składnia powyższa składnia jade odpowiadałaby:

```
<!DOCTYPE html>
<html>
<head>
     <title> ….. </title>
     <link rel="stylesheet" href="/stylesheets/style.css" />
</head>
<body>
    ……..
</body>
</html>
```

Jak widać składnia “semi-html” jade, jest pozbawiony znaków określających znaczniki oraz jest również samo uzupełniający się ponieważ nie ma znaczników zamykających!!!!
Przykład widoku wyświetlajacego stronę główną jest umieszczony w pliku “index.jade”:

```
extends layout

block content
  h1= title
  p Welcome to #{title}
```

Polecenie “extends layout” mówie serwerowi, że ten plik rozszerza funkcjonalność głównego widoku umieszczonego w pliku layout.

Dodatkowo w obu przypadkach widoku głównego, jak i widoku rozszerzającego widać również jak należy odbierać zmienne przekazywane przez w obiekcie funkcji render. Jak pamiętamy przekasywaliśmy w kontrolerze objekt:

```
{ title: ‘Express’}
```

Jest on teraz wyświetlany w dwóch miejscach w elemencie <h1> oraz <p>. W obu przypadkach dostęp do obiektu jest inny, jednak są to dwie równoważne metody. Z doświadczenie polecam metodę drugą czyli **`#{ nazwa zmiennej }`**.

**Zapisywanie znaczników w JADE:**

*Przykład 1. (który już znamy):*

| | |
|---|---|
| `<div>`<br>` Element blokowy`<br>`</div>` | `div`<br>`  \| Element blokowy` |

*Przykład 2.*

| | |
|---|---|
| `<html>`<br>`  <head>`<br>`  <title> Jade </title>`<br>`  </head>`<br>`  <body>`<br>`  <p>`<br>`  JADE`<br>`  </p>`<br>`  </body>`<br>`</html>` | `html`<br>`...head`<br>`.....title Jade`<br>`...body`<br>`.....p`<br>`.....| JADE` |

W przykładzie powyżej kropki oznaczają wcięcia zrobione za pomocą spacji. Alternatywnie można używać “tabulatora” (stałego odstępu). Nalezy jednak w pojedyńczym pliku .jade używać spacji lub tabulacji (nigdy obu). Powyższy przykład pokazuje również niezwykłą zwięzłość tego typu kodowania.

Wewnątrz tekstu języka JADE można używać znaczników HTML !!!!!

*Przykład 3.*

| | |
|---|---|
| `Plain text can include`<br>`<strong>html</strong>`<br>`<p>`<br>`  It must always be on its own line`<br>`</p>` | `| Plain text can include`<br>`<strong>html</strong>`<br>`p`<br>`  | It must always be on its own line` |

Znaczniki HTML zostaną przesłane do pprzegląderki gdzie zostaną zinterpretowa

*Przykład 4 - Lista*

| | |
|---|---|
| `<ul>`<br>`  <li>Item A</li>`<br>`  <li>Item B</li>`<br>`</ul>` | `ul`<br>`  li Item A`<br>`  li Item B`<br>`  li Item C` |

**Tworzenie atrybutów znaczników:**

| | |
|---|---|
| `<a href="google.com">Google</a>`<br>`<a href="google.com"`<br>`class="button">Google</a>` | `a(href='google.com') Google`<br>`a(class='button', href='google.com')`<br>`Google` |

**Klasy:**

| | |
|---|---|
| `<a class="button"></a>`<br>`<div class="content"></div>` | `a.button`<br>`.content` |

**Identyfikatory:**

| | |
|---|---|
| `<a id="main-link"></a>`<br>`<div id="content"></div>` | `a#main-link`<br>`#content` |

**Prosta logika w JADE:**

JADE umożliwia wykonywanie bardzo prostych operacji wyliczeniowych:

| | |
|---|---|
| `<ul>`<br>`  <li>1</li>`<br>`  <li>2</li>`<br>`  <li>3</li>`<br>`  <li>4</li>`<br>`  <li>5</li>`<br>`</ul>` | `ul`<br>`  each val in [1, 2, 3, 4, 5]`<br>`  li= val` |

Możliwe jest wykonywanie prostych zadań które mają być wykonane w pętli:

| | |
|---|---|
| `<li>item</li>`<br>`<li>item</li>`<br>`<li>item</li>` | `- for (var x = 0; x < 3; x++)`<br>`  li item` |

Dostęna jest również instrukcja warunkowa wyboru:

| | |
|---|---|
| `<p>you have 10 friends</p>` | `- var friends = 10`<br>`case friends`<br>`  when 0`<br>`  p you have no friends`<br>`  when 1`<br>`  p you have a friend`<br>`  default`<br>`  p you have #{friends} friends` |

***Przekazywanie zaminnych przez adres - metoda GET w node.js.***

W kilku sytuacjach niezbędne jest jawne przekazanie wartości zmiennych przez adres (metoda GET), podobanie jak to było w PHP. Jako, że w tym przpadku metoda wysyłamy zmienne to GET możemy posłużyć się funckja: “router.get()”.
Jak już wiemy w metodzie “.get()” pierwszym aprametrem jest adres obsługiwanego żądania. W tym miejscu możemy również przekazywać wartości zmiannych.

**Przykład:**
Chcemy przekazać 4 następujące zmienne o wartościach całkowitych: a=4, b=5, c=6, d=7. Aby to uczynić musimy w odpowiedni sposób spreparować maskę adresu dla metody “.get()”. Zakładamy że nasz funkcja będzie obsługiwała docelowo adres “/test” oraz w odpowiedzi będzie rederować strone widoku o nazie “testowy.jade”. W standardowym wypadku szablon naszej funkcji będzie następujący:

```
router.get('/test', function(req, res){
                var obj = 0;
                res.render('testowy', obj);
});
```

Teraz musimy tak zmodyfikowac maskę obsługiwanego adresu aby można było podać zmienne (oczywiście nie wpisujemy ich na sztywno w adresie!). Zmiane w adresie są symbolizowane przez:

```
:nazwa_zmiennej          // dwukropek i nazwa zmiennej
```

przykładowo:

```
router.get('/test/:a', function(req, res)
```

gdzie "**:a**" jest nazwą naszej zmiennej którą będziemy mogli posługiwać się do odebrania wartości. Jesmy mamy tych zmiennych wiecej oddzialamy je przecinkami:

```
 router.get('/test/:a,:b,:c,:d', function(req, res)
```

Możliwe są również, inne kombinacje np:

```
router.get('/test/:a/:b,:c/:d', function(req, res)
router.get('/test/:a/:b/:c/:d', function(req, res)
router.get('/test/:a/:b,:c,:d/:e/:f/:g,:h', function(req, res)
```

W adresach można przesyłać dowolne wartości tj. liczby, znaki etc.

Aby odebrać wartości ze zmiennych adresowych należy posługiwać się obiektem żadania i nazwą zmiennej zdefiniowanej w adresie:

```
req.params.nazwa_zmiannej;
```

np:

```
req.params.a;
req.params.b;
```

Wykorzystanie tej funkcjonalności najczęściej polega na wpisaniu na sztywno np. odnośnikach wartości zmiennych. W widoku ma to np następującą postać:

```
a(href="/test/4,5,6,7") kliknij mnie
```

Po kliknięciu w link zostanie wywołana metoda get obsługująca wywołany adres.

## ***Przekazywanie zaminnych przez frmularze - metoda POST w node.js.***

Zacznimy od zdefiniowania w "routes/index.js" nowej funkcji która będzie odpowiadała za wyświetlanie naszej strony oraz przetworzenie danych wpisanych do formularza.
Standardowo w tym pliku mamy objeekt który obsługuje nam zapytania do serwera żadające wyświetlanie danej strrony:

```
/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});
```

W tym wypadku w odpowiedzi renderowany jest szablon zapisany w pliku "views/index.jade", będacy rozszerzeniem domyślnego wyglądu strony. Aby sobłużyć formularz obsługujący dane musimy posłuzyć

się do tego celu metodą odbierającą dane z formularza czyli “POST” (wyjaśnienie różnicy pomiędzy GET a POST było przedstawione w poprzednich matariałach).
Szkielet funkcji jest następujący:

```
router.post( '[link-wyswietlany]', function(req,res){
    res.render(' [nazwa.jade] ',{ } );
});
```

W naszym przypadku chialibyśmy utworzyć formularz który posiada jedno pole typu “text” do którego wpisujemy dane, a następnie te dane są wyświetlane jak tytuł strony ( w nagłówku h1) strony wyświetlonej po przeładowaniu. Załużmy że strona po przeładowaniu będzie się nazywac “studentuj”, a do szablonu jade przekazywana będzie wartośc pola z formularza. Do dobierania danych z pól formularza, posługujemy się obiektem “req” który jest generowany w momencie wysyłania żadania do serwera. Dostęp do wszystkich pól formularza jest możliwy przez obiekt body i podanie nazwy danego pola w formularzu np:. req.body.student. Pełny kod funkcji renderującej stonę na podstawie danych z formularza jest następujący:

```
router.post('/studentuj',function(req,res) {
 var student = {
    name: req.body.student  //nazwa pola
   };
res.render('index', {title: req.body.student});
});
```

Musimy odpowiednio zmodyfikować plik “index.jade” tak aby zawierał sam formularz:

```
extends layout

block content
  h1= title
  p Welcome to #{title}
  form(method="POST" action="/studentuj")
    input(type="text" name="student")
    input(type="submit")
```

Formularz ten “POSTuje” (czyli wsysyła dane) pod adres “/studentuj” który własnie jest obsługiwany w kontrolerze przez wcześniej napisaną funkcję.

*Zadanie:*
*Proszę przymyśleć zadaną konstrukcję obsługi formularza w architekturze “View-Controller” i spróbować stworzyć projekt który odwzorowuje omówiony przykład wraz ze swoim autorskim formularzem.*

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**

(zajęcia 07.01.2016 r.)

Spróbujmy teraz dołączyć dla kompletności modelu trójwarstwowego “MODEL”. W tym celu należy stworzyć nowy projekt, a w jego w kartotece głownej utworzyć kartotekę “models”. W kartotece będą przechowywane moduły z różnymi autorskimi klasami np. do komunikacji z bazą danych. Narazie dla naszych potrzeb będzie prosta moduł przechowujący obiekt i z dwoma wartościami oraz zwracający dane do kontrolera za pomocą prostej metody:

```
module.exports = function Testowy(req){
// tworzymy obiekt, z dwoma polami
var obj = {
            "a":5,
            "b":9
      };
// tworzymy funkcję zwracjącą utworzony obiekt
      this.getMyObj = function(){
            return obj;
      }
};
```

Następnie musimy napisać funkcję kontrolera która obsłuży nam nasz nowy moduł. Na początek musimy dołączyć nasz moduł tak aby można było się nim posługiwać:

```
var Testowy = require('../models/test.js');
```

Następnie tworzymy funkcję która obsłuzy żądanie “GET” i wyświetli dane za pomoćą modułu “Testowy”:

```
router.get('/test', function(req, res) {
  var testowy = new Testowy(req);
  var obj = testowy.getMyObj();
  res.render('testowy',obj);
});
```

Chcemy aby nasze dane były renderowane pod adresem “test” dlatego musimy przygotować dla niego moduł jade np. “testowy.jade”:

```
extends layout

block content
  h1= name
    p Welcome to #{a}
      div
          span #{b}
```

Moduł ten odbiera dane z przesłane przez funkcję kontrolera, a odebrane przez kontroler z napisanego przez nas modułu.

*Zadanie:*

*Proszę utworzyć nowy projekt i zaimplementować na podstawie przykładu własny moduł wyświetlający dane z dowolnego zadeklarowanego obiektu.*

**ZAAWANSOWANE TECHNIKI WWW (WFAIS.IF-C112)**

(zajęcia 21.01.2016 r.)

Komunikacja bazodanowa w środowisku uruchomieniowym Node.js odbywa się dzięki interfojsowi pośredniczącemu w przesyłaniu danych pomiędzy aplikacją, a wybraną bazą danych. Interfejsy można podzialić na dwa typy: (a) wykorzystujących język SQL do wykonywania kwerend bazy danych, (b) wykorzystujących mapowanie obiektowo-relacyjne (ORM). Na ćwiczeniach będziemy wykorzystywać interfejs z punktu (a), natomiast na wykładzie omówione zostały również rozwiązania z punktu (b).

a. **Instalacja i interfejsu MySQL**

Do zainstalowania pakietu uzywamy standardowego narzędzia NPM:

```
npm instal mysql --save
```

Następnie uwzględniamy pakiet w wybranym projekcie w pliku controlera (routes/index.js):

```
var mysql = require('mysql');
```

b. **Ustanowienie połączenia z bazą danych**

Połączenie realizowane jest na podstawie podanych parametrów serwera bazodanowego: nazwy hosta, nazwy, użytkownika, hasła dostępu, numeru portu na którym serwer bazodanowy nasłuchuje połączeń. Określenie paraametrów komunikacyjnych odbywa się poprzez metodę .createConnection(), co ilustruje przykład poniżej:

```
var db = mysql.createConnection({
        host: 'localhost',
        user: 'user,
        password: 'naslo',
        database: 'nazwa_bazy'
});
```

Po określeniu parametrów połączenia należy zainicjalizować połączenie wykorzustując metodę .connect na obiekcie, który przechowuje informację o parametrach:

```
db.connect();
```

Od tego momentu mamy pełny dostęp do bazy danych i możliwosc wykonywania kwerdend SQL poprzez obiekt “db”.

### c. Wykonywanie kwerend SQL

W przypadku opisywanego interfejsu komunikacyjnego będziemy posługiwać się zapytaniami w języku SQL (SELECT, INSERT, UPDATE, DELETE etc.)
Każde zapytanie musi zostać zdefiniowane w postaci ciągu znakowego String np.

```
var zapytanie = 'SELECT * FROM tabela';
var zapytanie = 'SELECT * FROM tabela WHERE id=7';
```

Do wykonania zapytania i przesłania kwerendy do silnika bazodanowego służy metoda ".query()", która wykonujemy na obiekcie połączenia - w naszym przykładzie "db".
Metoda ta przyjmuje dwa parametry:

- obiekt zapytania (var zapytanie),
- funkcję wywołania zwrotnego która obsługuje rezultaty zwrócone z bazy danych function( error, dane) {}

Funkcja wywołania zwrotnego przyjmuje dwa obiekty:

- obiekt przechowujący informację o błędach komunikacji,
- obiekt w którym przechowywane są zwracane z bazy danych informacje. Obiekt ten jest obiektem typu Array (tablicowego).

Przykład użycia funkcji ".query()" dla bazy danych w której znajduje się relacja "users", z trzema polami: **`id (INT, PK, AI), login (VARCHAR), haslo (VARCHAR).`**

```
var sql = 'SELECT * FROM users';
  db.query(sql, function(error,dane){
        res.render('index', { title: 'Express', dane: dane});
   });
});
```

W przykładzie zwrócony obiekt "dane" jest następnie przekazany do widoku w standardowy sposób gdzie powinien zostać obsłużony po stronie widoku.

### d. Przekazywanie danych do widoku

Do widoku trafia obiekt "dane" który jest tablicą z polami których nazwy odpowiadają nazwą kolumn które znajdują się w bazie danych (patrz wyżej). Aby wyłuskać z tablicy pojedyńcze wiersze z obiektu "dane" najlepiej posłużyć się jedna z możliwych funkcji logicznych dostepnych po stronie widoku. W moim przykładzie skorzystam z funkcji "each":

```
extends layout

block content
  h1= title
  p Welcome to #{title}
  div
   ul
```

```
each item in dane
 li #{item.login} #{item.haslo}
```

W tym przykładzie obiekt “item” w pętli “each” reprezentuje pojedyńczy wiersz pozyskany z bazy danych. Aby dostać się do poszczególnych pól posługujemy się notacją obiektową (z kropką) gdzie nazwy własności obiektu są tożsame z nazwami kolumn w bazie danych.

***ZADANIE:***

*Proszę przygotować aplikacje w środowisku Express, która będzie:*

- *łączyła się z bazą danych korzystając z pakietu “mysql”,*
- *posiadała funcję router .get() pobierającą dane z tej bazy danych i wypisujące je w postaci tabelarycznej w widoku uzytkownika.*
- *posiadającą funkcje routera .post() obsługującą formularz, przez który będzie można zasilać bazę danych nowymi informacjami.*