1 Co to jest Java Script?

JavaScript jest językiem używanym do budowy stron internetowych. Nie zastępuje HTML, ale znacznie rozszerza jego możliwości. Jego głównym obszarem działania jest tworzenie i zarządzanie interakcją – a więc elementem, który w HTML sprowadzał się właściwie wyłącznie do możliwości kliknięcia i załadowania nowej strony.

JavaScript (JS) w przeciwieństwie do PHP jest językiem przetwarzanym w przeglądarce, a nie po stronie serwera.

Języka JavaScript nie należy utożsamiać z językiem Java, który jest od niego znacznie większą i poważniejszą platformą programistyczną.

JavaScript jest uważany za „lekki" język programowania.

2 Podstawy języka JavaScript.

Skrypty w języku JS mogą znajdować się w kodzie HTML (trochę w podobny sposób, jak kod PHP) albo być umieszczone w osobnym pliku tekstowym (któremu zwykle nadajemy rozszerzenie .js). Jeśli umieszczamy skrypt w pliku HTML, to stosujemy znacznik HTML <script>, przy czym niestety jego opis (i składnia) nieco różniła się w kolejnych wersjach języka HTML.

W dalszej części tego tutoriala będziemy posługiwać się prosta i wygodną postacią zalecaną dla HTML5. Nie występują tam żadne dodatkowe atrybuty, bowiem JS jest domyślnym językiem skryptowym dla HTML5.

```
<!DOCTYPE html>
<html>
<body>
  <script>
    // Tu będzie jakiś skrypt JavaScript
  </script>
</body>
</html>
```

2.1 Polecenia JS wykonywane przy wyświetlaniu strony

```
<!DOCTYPE html>
<html>
<body>
  <script>
    document.write("Hello world!");
  </script>
</body>
</html>
```

W tym przypadku wykorzystaliśmy funkcję write, która działa podobnie do PHP-owego echo, czyli wypisuje do przeglądarki określony tekst. Warto jednak zwrócić uwagę, że występuje przed nią coś jeszcze, a mianowicie konstrukcja „document."

Język JavaScript należy do tak zwanych języków obiektowych – nie wdając się w szczegółowe informacje programistyczne, oznacza to, że jeśli chcemy wykonać pewną funkcją, musimy o

to „poprosić” pewien obiekt. Obiektem tym w rozpatrywanym przypadku jest „document”, który symbolizuje całą stronę. Innymi słowy, polecenie document.write(„Hello world”) należy rozumieć tak: „drogi obiekcie DOKUMENT, wykonaj dla mnie funkcję WRITE z wartością „Hello world”.

Powyższy przykład zostanie wykonany w momencie, w którym przeglądarka przetwarza stronę (czyli ją wyświetla). Nie jest to jednak typowe użycie poleceń JS. Najważniejsze obszary działania skryptów JS to:

- modyfikowanie strony HTML, która już została wyświetlona (bez jej przeładowania!)
- reagowanie na dynamicznie pojawiające się zdarzenia

2.2 Skrypt JS modyfikujący już wyświetloną stronę

Skrypty JS działają w przeglądarce – więc są w stanie zmodyfikować treść, która jest wyświetlona. Popatrzmy na prosty przykład:

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8"></head>
<body>
  <p>To jest przykładowy akapit.</p>
</body>
</html>
```

To jest przykładowy akapit.

Dodajmy do niego (nic nie usuwając!) prosty skrypt JS i zobaczmy, jak będzie wyglądała strona wyświetlona w przeglądarce:

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8"></head>
<body>
  <p id="akapit">To jest przykładowy akapit.</p>
  <script>
     document.getElementById("akapit").innerHTML="Nowa wersja";
  </script>
</body>
</html>
```

Małe zaskoczenie – w przeglądarce ukazuje się jedynie „Nowa wersja”:

Nowa wersja

Przyjrzyjmy się, jak działa nasz skrypt.

Po pierwsze, zauważamy, że nasz akapit (znacznik <p>) wzbogaciliśmy o atrybut identyfikacji: <p id=”akapit”>. Pamiętajmy, że atrybut id w przeciwieństwie do atrybutu class jest unikalny. Tylko jeden obiekt w całym dokumencie HTML może mieć określoną wartość id.

Po drugie – rzut oka na skrypt. Jak widać, wykorzystaliśmy nową funkcję obiektu document. Jest nią getElementById. Służy ona do znajdowania określonego obiektu HTML (właśnie za pomocą id).

Zapis „document.getElementById("akapit").innerHTML” powinniśmy rozumieć jako: „drogi obiekcie DOCUMENT, znajdź mi proszę obiekt o identyfikatorze „akapit” a następnie jego

zawartość zamień na tekst „Nowa wersja”. InnerHTML to wszystko to, co znajduje się pomiędzy znacznikiem początkowym a odpowiadającym mu znacznikiem końcowym .

W tym miejscu jedna ważna uwaga. Jeśli zmienimy trochę kolejność naszych instrukcji (tak jak w przykładzie poniżej), to nasz skrypt nie zadziała – nadal będzie się pokazywało „To jest przykładowy akapit”.

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8"></head>
<body>
  <script>
     document.getElementById("akapit").innerHTML="Nowa wersja";
  </script>
  <p id="akapit">To jest przykładowy akapit.</p>
</body>
</html>
```

Dlaczego tak się dzieje? Odpowiedź jest stosunkowo prosta: przeglądarka wyświetla stronę WWW linia po linii. W związku z tym, gdy dochodzi do naszego skryptu (linie od <script> do </script>), to linia wyświetlająca akapit (To jest przykładowy akapit.) nie została jeszcze wyświetlona!

W związku z tym, funkcja getElementById("akapit") nie potrafi znaleźć obiektu o identyfikatorze id (bo go jeszcze nie ma!), a skoro tak – to nie może zmienić jego zawartości.

Dopiero gdy skrypt się skończył, przeglądarka dostaje polecenie wyświetlenia - ale skrypt już zadziałał.

To na pozór skomplikowane rozumowanie można porównać do procedury przygotowywania kawy:

- weź kubek

- nasyp kawy do kubka

- nalej wrzątku do kubka

Jeśli odwrócimy kolejność poleceń, to oczywiście nie zaparzymy naszej „małej czarnej”:

- nalej wrzątku do kubka (jakiego kubka!?!??!)

- weź kubek

- nasyp kawy do kubka

2.3 Polecenia JS wykonywane w reakcji na zdarzenia

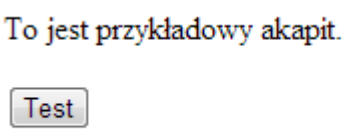
Główną zaletą skryptów JS jest możliwość określenia zachowania strony nie w momencie jej wyświetlenia, ale w momencie kiedy pojawia się określone zdarzenie (tzw. event). Zdarzeniem tym może być na przykład umieszczenie kursora myszki w określonym miejscu, kliknięcie elementu czy upływ pewnego czasu.

To właśnie obsługa zdarzeń jest najczęściej wykorzystywaną funkcjonalnością Javascript.

Wyobraźmy sobie prosty przykład – chcielibyśmy, aby po kliknięciu na przycisk, na stronie pokazywała się dodatkowa treść. Zacznijmy od przygotowania strony HTML z takim przyciskiem:

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8"></head>

<body>
  <p id="xxx">To jest przykładowy akapit.</p>
  <button type="button">Test</button>
</body>
</html>
```



Teraz przygotujmy kawałek kodu, który zastąpi zawartość (czyli innerHTML) obiektu o identyfikatorze xxx:

```
document.getElementById("xxx").innerHTML="Nowe!";
```

Żeby móc przywołać go w dowolnej chwili, wygodnie będzie zrobić z powyższego kodu tzw. własną funkcję. Wygląda to tak:

```
function mojeZmien () {
  document.getElementById("xxx").innerHTML="My First JavaScript Function";
}
```

Dzięki temu, w dowolnej chwili, kiedy będziemy chcieli wywołać nasz kod wystarczy po prostu wpisać mojeZmien(). Tworzenie własnych funkcji przypomina nieco zapamiętywanie zaklęć przez maga w grach RPG. Przygotowuje sobie on takie zaklęcie odpowiednio wcześniej – po to, by w dowolnej chwili z niego skorzystać. Funkcje w JS mają tu jednak wyższość – z raz przygotowanej funkcji możemy korzystać dowolnie wiele razy! :-)

Naszą świeżo zbudowaną funkcję umieścimy w sekcji <head>. Właściwie moglibyśmy umieścić ją – jak dotąd – w sekcji <body>, ale umieszczenie w <head> podkreśla, że to tylko „przygotowanie” do działania, a nie samo działanie:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script>
     function mojeZmien() {
       document.getElementById("xxx").innerHTML="Nowe!";
     }
  </script>
</head>

<body>
  <p id="demo">To jest przykładowy akapit.</p>
  <button type="button">Test</button>
</body>
</html>
```

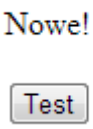
W tym miejscu ważne zastrzeżenie: w powyższym przykładzie, funkcja mojeZmien() nie wykonała się ANI RAZU! Ona jest tylko PRZYGOTOWANA do wykonania. To tak, jakbyśmy przygotowali sobie przepis na doskonałe ciasto. Przepis może być spisany i przyklejony do drzwi lodówki – ale zanim go nie „uruchomimy" – nie będziemy się mogli cieszyć udanym wypiekiem.

Spróbujmy więc spowodować, aby nasz „przepis" się wykonał:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script>
     function mojeZmien() {
       document.getElementById("xxx").innerHTML="Nowe!";
     }
  </script>
</head>

<body>
  <p id="xxx">To jest przykładowy akapit.</p>
  <button type="button">Test</button>
  <script>
    mojeZmien();
  </script>
</body>
```



Hmmm… wszystko ładnie – ale przecież chcieliśmy, żeby zmiana zawartości akapitu była widoczna nie w momencie załadowania strony, ale w momencie kliknięcia przycisku. Żeby tak się stało, musimy trochę nasz skrypt zmodyfikować i „nauczyć" przycisk <button>, że ma reagować wywołaniem funkcji mojeZmien() w momencie, gdy ktoś go naciśnie. W języku JS oznacza to wywołanie funkcji w momencie, gdy pojawi się zdarzenie „onclick":

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script>
     function mojeZmien() {
       document.getElementById("xxx").innerHTML="Nowe!";
     }
  </script>
</head>

<body>
  <p id="xxx">To jest przykładowy akapit.</p>
  <button type="button" onclick="mojeZmien()">Test</button>
</body>
</html>
```