Temat: Podstawy języka programowania Java Script.

JavaScript (w skrócie "JS") jest pełnoprawnym dynamicznym językiem programowania, który po dodaniu do dokumentu HTML, może dostarczyć dynamiczną zawartość do stron internetowych. Został stworzony przez Brendan'a Eich, współtwórcę projektu Mozilla, Mozilla Foundation i Mozilla Corporation.
JavaScript jest niezwykle wszechstronny. Zaczniemy z czymś małym, galeriami obrazków, zmiennymi układami strony i odpowiedziami na kliknięcia przycisków.

Warto zauważyć, że te cechy są wspólne dla wszystkich języków programowania.

Zmienne

Zmienne są kontenerami w których można zapisywać wartości. Zacznij od zadeklarowania zmiennej za pomocą słowa kluczowego var, a następnie dowolnej nazwy, której chcesz użyć:

```
var myVariable;
```

Średnik na końcu wiersza wskazuje, gdzie kończy się instrukcja; jest to bezwzględnie wymagane tylko w przypadku,
Możesz dowolnie nazwać zmienną, ale istnieją pewne zastrzeżone nazwy
JavaScript rozróżnia małe i duże litery — `myVariable`jest inną zmienną niż `myvariable`.
Po zadeklarowaniu zmiennej możesz nadać jej wartość:

```
myVariable = 'Bob';
```

Jeśli chcesz, możesz wykonać obydwie operacje w tej samej linii:

```
var myVariable = 'Bob';
```

Możesz pobrać wartość przez wywołanie zmiennej po nazwie:

```
myVariable;
```

Po podaniu wartości zmiennej można ją później zmienić:

```
var myVariable = 'Bob';
myVariable = 'Steve';
```

Warto zauważyć, że zmienne mają różne typy danych:

Typ	Wyjaśnienie	Przykład
String	Sekwencja tekstu znana jako ciąg znaków. Aby potwierdzić, że zmienna jest ciągiem, należy zamknąć jej wartość w apostrofach.	`var myVariable = 'Bob';`
Number	Liczba. Liczb nie zamyka się w apostrofach.	`var myVariable = 10;`

Typ	Wyjaśnienie	Przykład
Boolean	Prawda / Fałsz. Słowa `true` i `false` to specjalne słowa kluczowe w JS i nie potrzebują apostrofów.	`var myVariable = true;`
Array	Konstrukcja, która pozwala na przechowywanie wielu wartości w jednym odniesieniu.	`var myVariable = [1,'Bob','Steve',10];` Odwołaj się do każdego elementu tej tablicy: `myVariable[0]`, `myVariable[1]`, itd.
Object	Zasadniczo cokolwiek. Wszystko w JavaScript jest obiektem i może być przechowywane w zmiennej. Pamiętaj o tym podczas nauki.	`var myVariable = document.querySelector('h1');` Również wszystkie powyższe przykłady.

Więc dlaczego potrzebujemy zmiennych? Cóż, zmienne są potrzebne, aby zrobić cokolwiek interesującego w programowaniu. Jeśli nie moglibyśmy zmieniać wartości, to nie można by zrobić nic dynamicznego, jak personalizacja powitania lub zmiana wyświetlanego obrazu w galerii.

Komentarze

Możesz umieścić komentarze w kodzie JavaScript, tak samo jak w CSS:

```
/*
Wszystko pomiędzy to komentarz.
*/
```

Jeśli Twój komentarz nie zawiera przerw między wierszami, często łatwiej jest umieścić go za dwoma ukośnikami:

```
// To jest komentarz
```

Operatory

Operator jest symbolem matematycznym, który generuje wynik w oparciu o dwie wartości (lub zmienne). W poniższej tabeli można zobaczyć niektóre z najprostszych operatorów oraz kilka przykładów, które można wypróbować w konsoli JavaScript.

Operator	Wyjaśnienie	Symbole	Przykład
Dodawanie	Służy do dodawania dwóch liczb lub sklejenia dwóch ciągów znaków.	+	`6 + 9;` `"Hello " + "world!";`

Odejmowanie, Mnożenie, Dzielenie	Robią to, co można oczekiwać od nich w podstawowej matematyce.	-, *, /	`9 - 3;`
			`8 * 2; // mnożenie w JS jest gwiazdką`
			`9 / 3;`

Operator	Wyjaśnienie	Symbole	Przykład
Przypisanie wartości	Widzieliście już to: przypisuje wartość zmiennej.	`=`	`var myVariable = 'Bob';`
Znak równości	Wykonuje test sprawdzający, czy dwie wartości są sobie równe i zwraca wynik `true` / `false` (Boolean).	`===`	`var myVariable = 3;` `myVariable === 4;`

Zaprzeczenie, Nie równa się	Zwraca logicznie odwrotną wartość tego, co poprzedza; zmienia `true` w `false`, itd. Kiedy jest używany wraz z operatorem równości, operator negacji sprawdza, czy dwie wartości *nie* są równe.	`!`, `!==`	Podstawowe wyrażenie jest `true`, ale porównanie zwraca `false`, ponieważ zostało ono zanegowane:
			`var myVariable = 3;`
			`!(myVariable === 3);`
			Tu testujemy "czy `myVariable` NIE równa się 3". To zwraca wartość `false` ponieważ `myVariable` J EST równa 3.
			`var myVariable = 3;`
			`myVariable !== 3;`

Istnieje wiele więcej operatorów, ale to wystarczy na razie. Jeśli chcesz zobacz pełną listę sprawdź w Expressions and operators.

Mieszanie typów danych może powodować dziwne efekty podczas wykonywania obliczeń, dlatego należy uważać, aby prawidłowo odwoływać się do zmiennych i uzyskać spodziewane wyniki

Warunki

Warunkami są struktury kodu, które pozwalają na sprawdzenie, czy wyrażenie zwraca true, czy nie, i uruchamia inny kod ujawniony przez jego wynik. Bardzo popularną formą warunku są instrukcje if ... else. Na przykład:

```
var iceCream = 'chocolate';
if (iceCream === 'chocolate') {
  alert('Yay, I love chocolate ice cream!');
} else {
  alert('Awwww, but chocolate is my favorite...');
}
```

Wyrażenie wewnątrz if (...) jest testem — który używa operatora tożsamości (opisanego powyżej) w celu porównania zmiennej iceCream z ciągiem znaków chocolate, aby sprawdzić, czy te dwa są równe. Jeśli to porównanie zwróci true, uruchomiony zostanie pierwszy blok kodu. Jeśli porównanie nie jest prawdziwe, pierwszy blok jest pomijany, a drugi blok kodu, po wywołaniu else, jest uruchamiany.

Funkcje

Funkcje są sposobem na zapakowanie funkcjonalności, które chcesz wykorzystać ponownie. Gdy potrzebujesz procedury, zamiast pisać cały kod za każdym razem, możesz wywołać funkcję z nazwą funkcji. Powyżej widzieliście już niektóre zastosowania funkcji, na przykład:

```
1. var myVariable = document.querySelector('h1');
2. alert('hello!');
```

Funkcje te, document.querySelector i alert, są wbudowane w przeglądarkę, aby używać w dowolnym momencie.
Jeśli widzisz coś, co wygląda jak nazwa zmiennej, ale ma nawiasy — () — po niej, to prawdopodobnie jest to funkcja. Funkcje często biorą argumenty — bity danych potrzebne do wykonywania ich pracy. Znajdują się one w nawiasach, oddzielone przecinkami jeśli jest więcej niż jeden argument.
Na przykład, funkcja alert () powoduje pojawienie się okna podręcznego wewnątrz okna przeglądarki, ale musimy dać mu ciąg znaków jako argument, aby powiedzieć użytkownikowi o tym, co należy wyświetlić w wyskakującym okienku.
Dobrą wiadomością jest możliwość zdefiniowania własnych funkcji — w następnym przykładzie napiszemy prostą funkcję, która przyjmuje dwie liczby jako argumenty i mnoży je:

```
function multiply(num1,num2) {
  var result = num1 * num2;
  return result;
}
```

Spróbuj uruchomić powyższą funkcję w konsoli, a następnie przetestuj kilka argumentów. Na przykład:

```
multiply(4,7);
multiply(20,20);
multiply(0.5,3);
```

return informuje przeglądarkę o zwróceniu zmiennej result z funkcji, dzięki czemu jest ona dostępna. Jest to konieczne, ponieważ zmienne zdefiniowane wewnątrz funkcji są dostępne tylko w tych funkcjach

Zdarzenia

Prawdziwa interaktywność na stronie internetowej potrzebuje zdarzeń. Są to struktury kodu nasłuchające rzeczy, które dzieją się w przeglądarce i uruchamiajace kod w odpowiedzi.
Najbardziej oczywistym przykładem jest zdarzenie kliknięcia, które jest uruchamiane przez przeglądarkę po kliknięciu na coś za pomocą myszy. Aby to zademonstrować, wpisz następujący kod w konsoli, a następnie kliknij na bieżącej stronie internetowej:

```
document.querySelector('html').onclick = function() {
    alert('Ouch! Stop poking me!');
}
```

Istnieje wiele sposobów dołączania zdarzenia do elementu. Tutaj wybieramy element <html> i ustawiamy obsługę jego właściwości onclick równą funkcji anonimowej (tj. bezimiennej), która zawiera kod, który ma być uruchamiany.
Zauważ że

```
document.querySelector('html').onclick = function() {};
```

jest równe temu

```
var myHTML = document.querySelector('html');
myHTML.onclick = function() {};
```

To jest po prostu krócej.