JavaScript

4.0 परिचय

क्यों

JavaScript एक भाषांतरित स्क्रिप्ट की भाषा है. JavaScript कोड का प्रत्येक संग्रह कुछ और नहीं बल्कि रेखाओं का एक अनुक्रम है, जिसे एक अर्द्धविराम द्वारा समाप्त किया जाता है.. इसे . किसी JavaScript दस्तावेज़ में एम्बेड की गई स्क्रिप्ट की कोई भी संख्या हो सकती है.

JavaScript एक केस-संवेदी संबंधी भाषा है - जिसका अर्थ है कि इसे एकत्रित नहीं किया जा सकता (Java के विपरीत), इसे रनटाइम पर निष्पादित किया जाता है.

एक साथ रखी गई कई पंक्तियां स्क्रिप्ट बनाती हैं. कोड के एक खंड को मुड़े हुए कोष्ठकों {} द्वारा परिसीमित किया जाता है.

JavaScript और Java

यद्यपि JavaScript और Java में निकट का संबंध प्रतीत होता है, लेकिन वास्तव में वे बहुत अलग हैं. सबसे मूल भिन्नता यह है कि JavaScript ऑब्जेक्ट-उन्मुख सॉफ़्टवेयर विकास के प्रतिमान का समर्थन नहीं करती. Java एक दृढ़ता से लिखी जाने वाली भाषा है जबकि JavaScript चरों को गतिशील रूप से लिखा जाता है (इसके बारे में और जानकारी बाद में दी जाएगी). आवश्यक रूप से, JavaScript ऑब्जेक्ट गतिशील होते हैं - निष्पादन के दौरान किसी ऑब्जेक्ट के डेटा सदस्य और विधि बदल सकते हैं.

कैसे

JavaScript का इनलाइन, किसी <style> तत्व में या किसी अलग फ़ाइल में उपयोग किया जा सकता है.
हो सकता है कि आप अपना कोड इनलाइन लिखना चाहें, लेकिन यह बहुत अधिक उपयोगी नहीं होता क्योंकि आपको प्रत्येक बटन के लिए, संपूर्ण कोड पूरा लिखना होता है. स्क्रिप्ट लिखने का एक बेहतर तरीका हेडर टैग (`<head>...</head>`) में स्वतंत्र फ़ंक्शन के माध्यम से लिखना है. आप किसी अलग फ़ाइल में भी स्क्रिप्ट लिख सकते हैं.

इसे आज़माएं
इनलाइन स्क्रिप्ट का एक उदाहरण यहां दिया गया है. यह एक 'बटन' तत्व बनाता है और क्लिक किए जाने पर पॉप अप देता है.

```
<input type ="button" id="button1" value="Press me!"
onclick="alert('You have clicked the button!');" />
```

ऑनक्लिक इस बटन को क्लिक किए जाने पर की जाने वाली क्रिया को संदर्भित करता है.

> चेतावनी: इवेंट विशेषताओं का उपयोग नहीं किया जाना चाहिए. वे HTML को JavaScript के साथ मिला देती हैं, जिससे दस्तावेज़ पढ़ने और डीबग करने में कठिन और आकार में अधिक बड़े हो जाते हैं.

यहां आपके द्वारा JavaScript कोड को किसी शैली तत्व में लिखने का तरीका दिया गया है.

```
<head>
  <script>
              function doSomething() {
                              alert('You have clicked the button!') ;
              }
      var button1 = document.getElementById('button1');
      button1.onclicked = doSomething;
  </script>
</head>

<input type ="button" id="button1" value="Press me!" />
```

यह हमारे द्वारा ऊपर देखी गई कार्यात्मकता के समान है, सिवाय इसके कि यह कोड को लिखने का एक अधिक मॉड्यूलर तरीका है. बताने की आवश्यकता नहीं कि यह पसंदीदा विधि है!

4.1 JavaScript में तत्व पहुंच

क्यों
किसी XHTML फ़ॉर्म तत्व से संबंधित ऑब्जेक्ट के JavaScript में समाधान किए जाने के कई तरीके हैं. दस्तावेज़ ऑब्जेक्ट मॉडल (DOM) तरीका दस्तावेज़ ऑब्जेक्ट के फ़ॉर्म और तत्व सारणियों का उपयोग करना है, जिसे `Window` ऑब्जेक्ट द्वारा संदर्भित किया जाता है.
पुनः दोहराने के लिए, `Window` ऑब्जेक्ट वह विंडो है, जिसमें XHTML सामग्री प्रदर्शित की जाती है. प्रत्येक `Window` में दस्तावेज़ नामक प्रॉपर्टी होती है. प्रत्येक दस्तावेज़ ऑब्जेक्ट में एक फ़ॉर्म सारणी होती है और प्रत्येक तत्व दस्तावेज़ का कोई फ़ॉर्म दर्शाता है. फ़ॉर्म सारणी तत्व में प्रॉपर्टी के रूप में एक तत्व सारणी होती है जिसमें ऐसे ऑब्जेक्ट होते हैं जो XHTML फ़ॉर्म तत्व जैसे बटन, मेनू इत्यादि प्रदर्शित करते हैं.

कैसे

किसी दस्तावेज़ के तत्व डेटा और प्रचालन के साथ ऑब्जेक्ट होते हैं. डेटा को प्रॉपर्टी और प्रचालनों को विधियां कहा जाता है.
इस उदाहरण पर एक नज़र डालें:

```
<input type="text" name="age">
```

यहां, `type` और `name` एक ऑब्जेक्ट की प्रॉपर्टी हैं और 'text' और 'age' मान हैं.

DOM पता JavaScript ऑब्जेक्ट का पता है. आप डॉट प्रॉपर्टी का उपयोग करके DOM ट्री में और भीतरी तत्वों तक पहुंच सकते हैं. तत्व के नाम या अद्वितीय इसे संदर्भित करने में उपयोग किया जाता है. getElementByName() या getElementById() जैसी विधियों का उपयोग करके ऐसा किया जा सकता है.

इस उदाहरण पर एक नज़र डालें:

```
      var dom = document.getElementById("button1");
```

कोड की इस पंक्ति का उस तत्व तक पहुंचने में उपयोग किया जाता है जिसका अद्वितीय पहचान नाम या id *button1* है, जो दस्तावेज़ का एक भाग है.

इसे आज़माएं

<u>किसी फ़ॉर्म पर चिह्नित किए गए चेकबॉक की संख्या गिनने के लिए:</u>

यह स्क्रिप्ट चेकबॉक्स के माध्यम से 3 आइटम, की एक स्पष्ट सारणी बनाती है. सभी तत्वों का समान नाम और id होती है, जो उन्हें बाद में स्क्रिप्ट में पहुंचने में आसान बनाती है. तत्वों के अलग-अलग पतों को सारणी अनुक्रमणिकाओं का उपयोग करके संग्रहीत किया जाता है, जो 0 से प्रारंभ होकर -1 तक जाती हैं.

चूंकि इनपुट प्रकार चेकबॉक्स होता है, इसलिए प्रत्येक तत्व में भी 'चेक्ड'

. अगर बटन चेक किया गया है, तो चेक्ड प्रॉपर्टी true पर और अगर उसे चेक नहीं किया गया है तो false पर सेट की जाती है.

```
<form id="FavoriteColor">
  <input type="checkbox" name="color"
      value="red" /> Red
  <input type="checkbox" name="color"
      value="blue" /> Blue
  <input type="checkbox" name="color"
      value="green" /> Green
</form>


var numChecked = 0'
var dom = document.getElementById("FavoriteColor");
for (i = 0; i < dom.color.length; i++)
      if (dom.car[i].checked)
            numChecked++;
```

4.2 वैरिएबल, ऑपरेटर और टिप्पणियां

4.2.1 वैरिएबल

क्यों

सीधे शब्दों में कहें, तो वैरिएबल में मान होते हैं. किसी वैरिएबल का उपयोग करने से पहले, आपको उसे घोषित करना होगा.
JavaScript एक गतिशील रूप से लिखी जाने वाली भाषा है, जिसका अर्थ है कि किसी वैरिएबल का किसी भी काम में - किसी मान या किसी ऑब्जेक्ट के संदर्भ के लिए उपयोग किया जा सकता है.
अगर किसी वैरिएबल को कोई मान असाइन किया जाता है, तो यह अपने आप उस मान के प्रकार को ले लेता है.
JavaScript के प्रकार 2 समूहों में विभाजित किए जाते हैं:
प्रारंभिक प्रकार: संख्या, स्ट्रिंग, बूलियन, अनिर्धारित और शून्य
संदर्भ प्रकार: ऑब्जेक्ट, सरणियां और फ़ंक्शन

वैरिएबल के बारे में ध्यान देने योग्य कुछ बातें:

- JavaScript, HTML और CSS के विपरीत केस संवेदनशील है
- कीवर्ड (जैसे var, document इत्यादि) का वैरिएबल नामों के रूप में उपयोग नहीं किया जा सकता
- किसी वैरिएबल का डिफ़ॉल्ट मान (अगर अनिर्दिष्ट है) Undefined है
- स्ट्रिंग उद्धरणों के बीच घोषित की जानी चाहिए

किसी वैरिएबल का क्षेत्र:

- स्थानीय वैरिएबल:
  जब किसी फ़ंक्शन के अंदर कोई वैरिएबल घोषित किया जाता है, तो यह केवल उस फ़ंक्शन में पहुंच योग्य होता है और फ़ंक्शन के बंद हो जाने के बाद मान्य नहीं होगा.
- ग्लोबल वैरिएबल
  जब कोई वैरिएबल स्क्रिप्ट फ़ाइल के शीर्ष स्तर पर (किसी फ़ंक्शन के बाहर) घोषित किया जाता है, तब यह JavaScript प्रोग्राम में कहीं भी पहुंच योग्य होता है.
- अपरिभाषित वैरिएबल अपने आप ग्लोबल वैरिएबल होते हैं

कैसे

किसी वैरिएबल को या तो कोई मान असाइन करके (उस मामले में इंटरप्रेटर स्पष्ट रूप से इसका वैरिएबल होना घोषित करता है) या इसे ऐसे घोषणा कथन में सूचीबद्ध करके जो `var` कीवर्ड से शुरू होता है, कोई मान आसाइन किया जा सकता है.

इसे आज़माएं

```
वैरिएबल काउंटर,
        pi = 3.14,
        flag = true,                    // बूलियन वैरिएबल
        shape = “circle";           //स्ट्रिंग वैरिएबल
```

4.2.2 ऑपरेटर

क्यों

ऐसे कई फ़ंक्शन होते हैं, जिन्हें मानों पर निष्पादित करना होता है जैसे जोड़, घटाव इत्यादि. ये फ़ंक्शन ऑपरेटर

का उपयोग करके निष्पादित किए जाते हैं. कुछ ऑपरेटर जैसे + का संयोजन वाली स्ट्रिंग पर भी उपयोग किया जा सकता है.

कैसे

JavaScript में विभिन्न प्रकार के ऑपरेटर होते हैं.

- अंकगणित
  - यह मूल अंकगणितीय फ़ंक्शन करता है.

+	जोड़
-	घटाव
*	गुणा
/	भाग
%	शेषफल फ़ंक्शन
++,--	किसी वैरिएबल में वृद्धि/कमी (एकांगी ऑपरेटर)

+ concat ऑपरेटर के रूप में भी उपयोग किया जा सकता है.

```
var a = "Hello" + "World"
```

इसका अर्थ है कि *Hello World* .

- समानता
  - इसका उपयोग 2 मानों और प्रकारों की तुलना करने में किया जाता है

=	असाइनमेंट ऑपरेटर `var a = 2` वैरिएबल a को मान 2 असाइन करता है

==	समानता ऑपरेटर
	`var a = "2" == 2;`
	//true
	यह दो वैरिएबल के प्रकारों और मानों की जांच करता है

- तुलना
  - इसका उपयोग 2 मानों की तुलना करने में किया जाता है. इसका उपयोग स्ट्रिंग की तुलना करने में भी किया जा सकता है.

< , >	इससे कम, इससे अधिक
<= , >=	इससे कम या इसके बराबर, इससे अधिक या इसके बराबर

4.2.3 टिप्पणियां

क्यों

टिप्पणियां किसी कोड में मानव द्वारा पढ़ी जा सकने वाली पंक्तियां होती हैं, जिन्हें कंपाइलर द्वारा निष्पादित नहीं किया जाता.
उन्हें किसी विशिष्ट प्रारूप का उपयोग करने की आवश्यकता नहीं होती. इसे कोड में केवल प्रोग्रामर की इस बात में मदद करने के लिए जोड़ा जाता है कि कोड के खंड का क्या उद्देश्य है.

कैसे

एकल पंक्ति वाली पंक्तियां // के साथ लिखी जानी चाहिए
बहु-पंक्ति वाली टिप्पणियां /* और */ में लिखी जानी चाहिए

इसे आज़माएं

var a = 2 ; //यह एक टिप्पणी है. यह एकल पंक्ति में होनी चाहिए.
var b = “Hello" ; /*यह एक बहु-पंक्ति टिप्पणी है.
इसे एक से अधिक पंक्ति में लिखा जा सकता है.*/

4.3 फ़ंक्शन, ऑब्जेक्ट और इवेंट

4.3.1 फ़ंक्शन

क्यों

फ़ंक्शन कोड का एक स्वतंत्र मॉड्यूल होता है जो कोई विशिष्ट क्रिया करता है.
यह आपके कोड की पुनः उपयोगिता को बेहतर बनाता है, विशेषकर जब आपको कोई विशेष क्रिया बार-बार करनी हो जैसे जोड़ना, उपयोगकर्ता इनपुट मांगना इत्यादि.

कैसे

फ़ंक्शन की परिभाषा फ़ंक्शन और के रूप में जाने जाने वाले मिश्रित कथन से मिलकर बनती है.
फ़ंक्शन हेडर आरक्षित शब्द (कीवर्ड) फ़ंक्शन . इसका एक अद्वितीय नाम भी होता है जो फ़ंक्शन का उद्देश्य बताता है. मुख्य भाग में कथन होते हैं जो उसके बाद फ़ंक्शन द्वारा निर्धारित की गई क्रियाएं करते हैं.
JavaScript फ़ंक्शन ऑब्जेक्ट होते हैं, इसलिए उनका संदर्भ देने वाले वैरिएबल को अन्य ऑब्जेक्ट संदर्भों के समान माना जा सकता है.
फ़ंक्शन में पैरामीटर भी हो सकते हैं. पैरामीटर किसी फ़ंक्शन हेडर में आने वाले वैरिएबल होते हैं जिनमें मान हो सकते हैं. जब किसी फ़ंक्शन को कॉल किया जाता है, तो फ़ंक्शन में पास किए जा सकते हैं जिसका अर्थ है कि मानों को इनपुट के रूप में फ़ंक्शन में पास किया जा सकता है. इसे फिर फ़ंक्शन के भीतर उपयोग किया जा सकता है. एक महत्वपूर्ण ध्यान देने वाली बात यह है कि फ़ंक्शन पैरामीटर कॉल-बाय वैल्यू विधि का उपयोग करके पास किए जाते हैं. इसका अर्थ है कि केवल मान को (और न कि ऑब्जेक्ट के संदर्भ को) इनपुट के रूप में फ़ंक्शन में पास

किया जा सकता है.

```
function functionName(var1, var2) {
              /* function specific code here
      …
              */
              Return returnVal; // optional return value
}
```

इसे आज़माएं
उपयोगकर्ता से 2 स्ट्रिंग स्वीकार करने के लिए एक फ़ंक्शन लिखें और फिर इसे एक संयोजित स्ट्रिंग के रूप में लौटाएं.

```
function returnConcatString(var1 var2) {
              var result = "";
//initialising a blank string for output
       for (var i = 0; i < arguments.length; i+ +) {
                              result = result + " " + arguments[i];
              }
      return result;
}

var b = returnConcatString("Hello", "World");
```

b में Hello World .

4.3.2 ऑब्जेक्ट

क्यों
एक ऑब्जेक्ट प्रॉपर्टी का एक संग्रह होता है. अगर किसी प्रॉपर्टी का मान कोई फ़ंक्शन है, तब इसे विधि कहा जाता है.
एक ऑब्जेक्ट में आवश्यक रूप से 2 चीज़ें हो सकती हैं: प्रॉपर्टी और विधि.
इसे और आसानी से बताने के लिए, एक जैसे सभी ऑब्जेक्ट में समान प्रॉपर्टी होती हैं लेकिन उनके प्रॉपर्टी मान भिन्न हो सकते हैं.

कैसे
एक ऑब्जेक्ट इससे संबंधित प्रॉपर्टी और विधियों को परिभाषित करके बनाया जाता है. आप डॉट ऑपरेटर (.) का उपयोग करके प्रॉपर्टी और विधियों को एक्सेस कर सकते हैं
किसी ऑब्जेक्ट की प्रॉपर्टी को इस तरह एक्सेस किया जा सकता है:

```
       objectName.propertyName ;
              objectName["propertyName"] ;
```

किसी प्रॉपर्टी की विधियों को इस तरह एक्सेस किया जा सकता है:

```
       objectName.methodName() ;
```

किसी रिक्त ऑब्जेक्ट को कंस्ट्रक्टर का उपयोग करके इस तरह घोषित किया जा सकता है:

```
var car = new Object () ;
```

ऑब्जेक्ट बनाने के बाद, प्रॉपर्टी को आरंभ किया जा सकता है.

```
car.type = "Sedan";
car.color = "red";
```

इसे आज़माएं

विविध प्रॉपर्टी जैसे प्रकार, रंग और मॉडल के साथ कार के प्रकार का कोई ऑब्जेक्ट बनाएं.

```
var car = {type: 'Sedan',
                              color: 'red',
                              Model: 'Fiat};
```

4.3.3 इवेंट

क्यों

इवेंट एक उपयोगकर्ता कार्रवाई या इस बात का नोटिफ़िकेशन है कि कुछ हुआ है. JavaScript कोड इवेंट के माध्यम से HTML पृष्ठों से इंटरैक्ट करता है. उदाहरण के लिए,किसी बटन को क्लिक करना, अपने माउस को किसी लेख पर ले जाना इत्यादि सभीइवेंट के उदाहरण हैं.

कैसे

इवेंट हैंडलर एक स्क्रिप्ट है जिसे किसी इवेंट के होने पर स्पष्ट रूप से कॉल किया जाता है. प्रत्येक तत्व में एक संबद्ध इवेंट विशेषता होती है. कई तत्वों में एकसमान इवेंट विशेषताएं भी हो सकती हैं.
इसका अर्थ है कि किसी इवेंट जैसे `onclick` में किसी लिंक (एंकर टैग) और पाठ बॉक्स में समान क्रियाएं होंगी.

यहां कुछ इवेंट और उनकी टैग विशेषताएं दी गई हैं:

इवेंट	टैग विशेषता
blur	onblur
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeypress
load	onload

mousedown	onmousedown
mousemove	onmousemove
mouseover	onmouseover
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

इसे आज़माएं
दस्तावेज़ के मुख्य भाग के लोड हो जाने पर एक सूचना संदेश भेजें.

```
<html>
<head>
              <title> load.html </title>
              <script type = "text/javascript"  src = "load.js">
              </script>
</head>
<body onload = load_greeting();">
              <p> पृष्ठ लोड हो गया है! </p>
</body>
</html>

//load.js
function load_greeting() {
              alert("This is a pop up message with a greeting - Hi!
Welcome to my page!");
}
```

4.4 स्ट्रिंग, संख्याएं, गणित और दिनांक फ़ंक्शन

4.4.1 स्ट्रिंग

क्यों
JavaScript स्ट्रिंग को संग्रहीत नहीं किया जाता है, न ही उन्हें वर्णों की सारणी माना जाता है, इसकी बजाय वे इकाई अदिश मान होते हैं.

कैसे
स्ट्रिंग में किए जा सकने वाले कई फ़ंक्शन या प्रचालन हो सकते हैं.

स्ट्रिंग विधियों का इस तरह से स्ट्रिंग प्रारंभिक मानों के माध्यम से उपयोग किया जा सकता है, जैसे कि मान ऑब्जेक्ट थे.
उदाहरण के लिए, किसी स्ट्रिंग में वर्णों की संख्या `length` प्रॉपर्टी में संग्रहीत होती है.

कुछ सामान्य स्ट्रिंग विधियां नीचे तालिका में दिखाई गई हैं.

विधि	पैरामीटर	परिणाम
length	कुछ नहीं	स्ट्रिंग ऑब्जेक्ट में वर्णों की संख्या लौटाता है
charAt	एक संख्या	निर्दिष्ट स्थान पर स्ट्रिंग ऑब्जेक्ट में वर्ण लौटाता है
indexOf	एक-वर्ण स्ट्रिंग	स्ट्रिंग ऑब्जेक्ट में पैरामीटर का स्थान लौटाता है
substring	दो संख्याएं	पहले पैरामीटर स्थान से दूसरे पैरामीटर पर स्ट्रिंग ऑब्जेक्ट की सबस्ट्रिंग लौटाता है
toLowerCase	कोई नहीं	किसी भी बड़े अक्षर से छोटे अक्षर में बदलता है
toUpperCase	कोई नहीं	किसी भी छोटे अक्षर को बड़े अक्षर में बदलता है

इन विधियों के अतिरिक्त, एस्केप वर्ण कहे जाने वाले कुछ वर्ण अनुक्रमों का भी उपयोग किया जाता है.

\n	नई पंक्ति
\t	टैब (8 रिक्त स्थान)
\r	कैरिज रिटर्न

इसे आज़माएं

```
var str = "Hello World" ;

str.charAt(2);                    // आउटपुट l है
str.indexOf('o');                 // आउटपुट 4 है
str.substring(2,4);               // आउटपुट llo है
str.toLowerCase();                // आउटपुट hello world है
```

4.4.2 संख्याएं

क्यों

संख्या ऑब्जेक्ट में उपयोगी प्रॉपर्टी का संग्रह है जिसमें स्थिर मान हैं.

कैसे

निम्न तालिका संख्या प्रॉपर्टी दर्शाती है.

प्रॉपर्टी	मतलब
MAX_VALUE	सबसे बड़ी प्रदर्शनीय संख्या
MIN_VALUE	सबसे छोटी प्रदर्शनीय संख्या
NaN	संख्या नहीं है
POSITIVE_INFINITY	अनंत संख्या दर्शाने वाला विशेष मान
NEGATIVE_INFINITY	ऋणात्मक अनंत संख्या दर्शाने वाला विशेष मान
PI	pi (3.14) का मान

4.4.3 गणित फ़ंक्शन

क्यों

Math ऑब्जेक्ट में त्रिकोणमितीय फ़ंक्शन जैसे कि ज्या और कोज्या के लिए विधियां दी गई हैं. इसमें अन्य सामान्य रूप से उपयोग किए जाने वाले गणितीय फ़ंक्शन के लिए विधियां भी हैं.

कैसे

गणित फ़ंक्शन में sin (sine) और cos (cosine) जैसे त्रिकोणमितीय फ़ंक्शन और साथ ही `floor`, जो किसी संख्या को छोटा कर देते हैं; `round`, जो संख्या को निकटतम पूर्णांक बना देते हैं; `max`, जो दो संख्याओं में से सबसे बड़ी संख्या लौटाता है आदि जैसे विशेष गणितीय फ़ंक्शन हैं.
सभी Math फ़ंक्शन का संदर्भ Math ऑब्जेक्ट का उपयोग करके दिया गया है.

इसे आज़माएं

```
Math.floor(4.2343);                    //आउटपुट 4 है
```

4.4.4 दिनांक फ़ंक्शन

क्यों

Date फ़ंक्शन का उपयोग दिनांक और समय को दर्शाने के लिए किया जाता है.

कैसे

Date ऑब्जेक्ट का संदर्भ Date ऑब्जेक्ट का उपयोग करके दिया जाता है.
इसे इस कथन का उपयोग करके बनाया जा सकता है:

```
var today = new Date() ;
```

यहां Date ऑब्जेक्ट की कुछ सामान्य रूप से उपयोग की गई विधियां दी गई हैं.

विधि	लौटाता है
toLocaleString	दिनांक जानकारी की स्ट्रिंग
getDate	महीने का दिन
getMonth	वर्ष का महीना (0 से 11 तक)
getDay	सप्ताह का दिन (0 से 6 तक)
getFullYear	यह वर्ष
getTime	01-01-1970 के बाद से मिलीसेकंड में संख्या
getHours	घंटे की संख्या (0 से 23 तक)
getMinutes	मिनट की संख्या (0 से 59 तक)
getSeconds	सेकंड की संख्या (0 से 59 तक)

4.5 सरणियां

क्यों

सारणी मानों का एक संग्रह है, जिसका संदर्भ उसी नाम से दिया जाता है. प्रत्येक सारणी तत्व को उसकी अनुक्रमणिका से एक्सेस किया जाता है. अनुक्रमणिका 0 से लंबाई-1 तक होती है.
अन्य प्रोग्रामिंग सारणी के विपरीत, JavaScript में ध्यान देने वाली महत्वपूर्ण बात यह होती है कि सरणियों में मिश्रित प्रकार हो सकते हैं. इसका मतलब यह है कि किसी सारणी के सभी तत्वों का एक ही प्रकार का होना आवश्यक नहीं है.

कैसे

Array ऑब्जेक्ट, अधिकांश अन्य JavaScript ऑब्जेक्ट के विपरीत, दो तरीके से बनाए जा सकते हैं.
पहली विधि किसी ऑब्जेक्ट को बनाने की सामान्य विधि है, जिसमें ऑपरेटर का उपयोग किया जाता है.

```
var list = new Array (1, 2, “three", “fo ur") ;
```

दूसरी विधि है शाब्दिक सारणी मान का उपयोग करना.

```
var list_2 = [1, 2, “three", “four"];
```

किसी भी सारणी मान की निम्नतम सूचकांक शून्य है.
Array तत्वों को अनुक्रमणिका के संख्यात्मक मान का सारणी नाम के सबस्क्रिप्ट के रूप में उपयोग करके ऐक्सेस किया जाता है.
सारणी की लंबाई किसी सारणी के तत्वों की संख्या होती है. इसलिए, किसी सारणी की अंतिम सूचकांक हमेशा लंबाई-1 रहेगी.

सरणियों के साथ कई विधियां संबद्ध होती हैं. कुछ सामान्य नीचे सूचीबद्ध हैं.

विधि	उद्देश्य
join	किसी सारणी ऑब्जेक्ट के सभी तत्वों को स्ट्रिंग में बदल देता है और उन्हें शृंखला में लिंक कर देता है
reverse	सारणी ऑब्जेक्ट के तत्वों का क्रम उल्टा कर देता है
sort	किसी सारणी ऑब्जेक्च के सभी तत्वों को बदलता है और उन्हें वर्णानुक्रम में क्रमबद्ध करता है
concat	concat विधि के वास्तविक पैरामीटर सारणी ऑब्जेक्ट के अंत में जोड़ता है
slice	सारणी ऑब्जेक्ट का वह भाग लौटाता है जिसे पैरामीटर के रूप में निर्दिष्ट किया गया है
pop	किसी सारणी ऑब्जेक्ट के उच्च अंत से तत्व निकालता है
push	किसी सारणी ऑब्जेक्ट के उच्च अंत में तत्व जोड़ता है

इसे आज़माएं
कोई सारणी ऑब्जेक्ट बनाएं और मान प्रदर्शित करें. सारणी में और तत्व जोड़ें और नए मान प्रदर्शित करें.

```
//array_example.js
var list = [2, 4, 6];

for (var i = 0; i < list.length; i++) {
              document.write(list[i] + "  ");
}
document.write("<br />");

var new_list = list.concat(8, 10);

for (var j = 0; j < new_list.length; j++) {
              document.write(new_list[j] + "  ");
}
```

आउटपुट निम्न होगा:
2 4 6
2 4 6 8 10

4.6 बूलियन, शर्तें और लूप

4.6.1 बूलियन

क्यों

Boolean JavaScript में एक डेटा प्रकार है. इसमें केवल 2 मान हो सकते हैं:

True और False.

कैसे

बूलियन ऑब्जेक्ट को JavaScript में किसी भी अन्य ऑब्जेक्ट की तरह घोषित किया जाता है.

```
var bool_t = new Boolean() ;
        bool_t = true;
```

4.6.2 शर्तें

क्यों

शर्त कथन इसलिए महत्वपूर्ण होते हैं क्योंकि वे कथनों के निष्पादन की प्रवाह निर्धारित करते हैं. वे हमेशा ऐसे पैरामीटर लेते हैं, जो बूलियन मान लौटाते हैं. अगर बूलियन मान सही है, तो शर्त के बाद तुरंत कथनों का खंड निष्पादित कर दिया जाता है.
वे तार्किक ऑपरेटर पर निर्भर करते हैं.

कैसे

वे अभिव्यक्तियां जिनपर कथन प्रवाह नियंत्रण आधारित हो सकते हैं, उनमें प्राथमिक मान, संबंधात्मक अभिव्यक्तियां और मिश्रित अभिव्यक्तियां शामिल हैं. नियंत्रण अभिव्यक्ति का परिणाम या तो True या False होता है.
JavaScript में AND, OR और NOT बूलियन ऑपरेशन के भी ऑपरेटर होते हैं. ये && (AND), || (OR) and ! (NOT) हैं.

शर्त कथन दो प्रकार के होते हैं:

1. चयन कथन

   यह if-then और if-then-else निर्माण पर निर्भर करता है.
   सिंटैक्स निम्न प्रकार है:

```
if (condition) {
                    /* अगर शर्त का मूल्यांकन परिणाम सही दिखाता है,
     तो इन कथनों को निष्पादित करें
…
                    */  }
                    else {
                    /* अगर शर्त का मूल्यांकन परिणाम गलत दिखाता है,
     तो इन कथनों को निष्पादित करें
…
                    */  }
```

2. switch कथन

   इसका उपयोग संभवतः तब किया जा सकता है, जब जांच किए जाने के लिए कई शर्तें हों और केवल एक

का मिलान हो सकता है. सिंटैक्स निम्न प्रकार है:

```
switch (expression) {
                                case value_1:
  //statement(s)
                                case value_2:
  //statement(s)
      ...
      case value_n:
//statement(s)
      [default:                          //statement(s)]
}
```

switch निर्माण की अभिव्यक्ति का मूल्यांकन किया जाता है और मान की तुलना निर्माण के केस के मानों से की जाती है. अगर एक का मिलान होता है, तो नियंत्रण को केस मान के बाद कथनों में ट्रांसफ़र कर दिया जाता है. निष्पादन तब शेष निर्माण के लिए जारी रहता है.
अधिकांश मामलों में, केवल एक केस कथन को निष्पादित करने की आवश्यकता होती है, जिसके बाद नियंत्रण को तुरंत कथनों को switch निर्माण के बाहर ट्रांसफ़र करना होगा.
उस स्थिति में, *break* कथन का उपयोग किया जा सकता है.
break कथन नियंत्रण को उस मिश्रित कथन से बाहर ट्रांसफ़र कर देता है, जिसमें वह दिखाई देता है.

इसे आज़माएं
दो संख्याओं की तुलना करने के लिए एक सरल if-else निर्माण लिखें और बड़ी संख्या आउटपुट करें.

```
//if_then.js

var a = 5;
var b = 8;

if (a > b)
   document.write("a, b से बड़ा है <br/> ");
else
   document.write("a, b से छोटा है <br/> ");
```

4.6.3 लूप

क्यों
लूप का उपयोग किसी सारणी या सूची में प्राथमिक रूप से उसी कार्य को कई तत्वों पर करने के लिए किया जाता है. मूल रूप से, यह कई बार कोड के एक ब्लॉक के माध्यम से लूप करता है.

कैसे
लूप में आमतौर पर नियंत्रण अभिव्यक्ति होता है, जिसके बाद कोष्टक में कोड का ब्लॉक होता है. लूप दो प्रकार के होते हैं:

1. While लूप

While लूप तब तक कोड की पंक्तियों का सेट निष्पादित करता रहता है, जब तक कि नियंत्रण अभिव्यक्ति अब सही नहीं रहती. सिंटैक्स निम्न प्रकार है:

```
while (conditional statement)
        {
                        /* कथनों का सेट
    …
                        */
        }
```

2. For लूप

For लूप में एक आरंभिक मान, एक सशर्त मान और एक वृद्धि चरण होता है. हर बार कथनों का ब्लॉक निष्पादित होने पर वैरिएबल का आरंभिक मान बढ़ता रहता है. जब सशर्त मान आगे सही नहीं रहता है, तब लूप समाप्त हो जाता है. सिंटैक्स निम्न प्रकार है:

```
for (initial expression; control express ion; increment
expression)
    {
                        /* कथनों का सेट
    …
                        */
        }
```

दोनों लूप को break कथन का उपयोग करके मांग पर समाप्त किया जा सकता है.

इसे आज़माएं

संख्या 1 से 50 तक का उत्पाद ढूंढने के लिए कोई JavaScript कोड लिखें.

```
//loops_while.js

var i = 1;
var sum = 0;

while ( i <= 50) {
sum = sum * i ;
}

document.write("उत्पाद" + sum); है

//loops_for.js
var sum = 0;
for (var i = 0; i < 51; i++)
sum = sum * i;
```

`document.write(“उत्पाद" + sum);` है

4.7 रेगुलर एक्सप्रेशन (RegEx)

क्यों

JavaScript में रेगुलर एक्सप्रेशन पर आधारित क्षमताओं से मिलान करने वाला शक्तिशाली प्रतिमान है. हालांकि स्ट्रिंग ऑब्जेक्ट के `search` फ़ंक्शन का उपयोग किया जा सकता है, फिर भी RegEx अधिक व्यापक कार्यक्षमता प्रदान करता है.

कैसे

'सामान्य' वर्ण वे होते हैं जो मेटा-वर्ण नहीं होते हैं, यानी, स्थिति के अनुसार उनके विशेष अर्थ नहीं होते हैं. विशेष वर्णों या मेटा-वर्णों में निम्न शामिल हैं:

```
\ | ( ) [ ] { } ^ $ * + ? .
```

मेटा-वर्णों के ठीक आगे बैकस्लैश (\) लगाकर उसका रेगुलर एक्सप्रेशन से मिला किया जाता है.

विरामचिह्न (.) एक नई पंक्ति को छोड़कर किसी भी वर्ण से मेल खाता है. जैसा ऊपर बताया गया है, किसी स्ट्रिंग में विरामचिह्न से मिलान करने के लिए, उसके आगे बैकस्लैश लगाना चाहिए.
उदाहरणः
`/snow./` का मिलान “snowy", “snowed", “snows" आदि से होगा

अगर किसी रेगुलर एक्सप्रेशन में सिकमफ़्लक्स वर्ण (^) पहला वर्ण है, तो वह निर्दिष्ट सेट को नकार देता है या उसे उलटा कर देता है.
उदाहरणः
[^abc] का मिलान 'a', 'b' और 'c' को छोड़कर सभी वर्णों से होगा

हाइफ़न (-) सीमा निर्दिष्ट करता है.

निम्न तालिका पूर्वनिर्धारित वर्गों के बारे में अधिक जानकारी देती है

नाम	समतुल्य प्रतिमान	मिलान
`\d`	`[0-9]`	अंक
`\D`	`[^0-9]`	अंक नहीं
`\w`	`[A-Za-z_0-9]`	अक्षरांकीय वर्ण
`\W`	`[^A-Za-z_0-9]`	अक्षरांकीय अंक नहीं
`\s`	`[ \r\t\n\f]`	वाइटस्पेस वर्ण
`\S`	`[^ \r\t\n\f]`	वाइटस्पेस वर्ण नहीं

इसे आज़माएं

[\d\.\d\d] //किसी अंक से मिलान करता है, जिसके बाद विराम-चिह्न और फिर 2 अंक आते हैं
[\D\d\D] //एकल अंक से मिलान करता है
[\d+\.\d*] //एक या उससे अधिक अंकों से मिलान करता है, जिसके बाद विराम-चिह्न और फिर 0 या अधिक अंक होते हैं
[\w\w\w] //3 निकटवर्ती शब्द वर्णों से मिलान करता है