Java Script tutorials – Εργαστηριακό μάθημα 1.

Η ετικέτα "script"

Τα προγράμματα JavaScript μπορούν να εισαχθούν σε οποιοδήποτε μέρος ενός εγγράφου HTML με τη βοήθεια της `<script>`ετικέτας.

Για παράδειγμα:

```
<!DOCTYPE HTML>
<html>

<body>

  <p>Before the script...</p>

  <script>
    alert( 'Hello, world!' );
  </script>

  <p>...After the script.</p>

</body>

</html>
```

Παλιά σήμανση

Η `<script>`ετικέτα έχει μερικά χαρακτηριστικά που σπάνια χρησιμοποιούνται στις μέρες μας αλλά εξακολουθούν να βρίσκονται στον παλιό κώδικα:

Το `type` χαρακτηριστικό: `<script type=…>`

Το παλιό πρότυπο HTML, HTML4, απαιτούσε ένα σενάριο για να έχει ένα `type`. Συνήθως ήταν `type="text/javascript"`. Δεν απαιτείται πια. Επίσης, το σύγχρονο πρότυπο HTML άλλαξε εντελώς την έννοια αυτού του χαρακτηριστικού. Τώρα, μπορεί να χρησιμοποιηθεί για λειτουργικές μονάδες JavaScript. .

Το `language` χαρακτηριστικό:`<script language=…>`

Αυτό το χαρακτηριστικό προοριζόταν να δείξει τη γλώσσα του σεναρίου. Αυτό το χαρακτηριστικό δεν έχει πλέον νόημα, επειδή η JavaScript είναι η προεπιλεγμένη γλώσσα. Δεν χρειάζεται να το χρησιμοποιήσετε.

Σχόλια πριν και μετά σενάρια.

Σε πραγματικά παλιά βιβλία και οδηγούς, μπορεί να βρείτε σχόλια μέσα σε `<script>`ετικέτες, όπως αυτό:

```
<script type="text/javascript"><!--
    ...
//--></script>
```

Αυτό το τέχνασμα δεν χρησιμοποιείται στη σύγχρονη JavaScript. Αυτά τα σχόλια κρύβουν κώδικα JavaScript από παλιά προγράμματα περιήγησης που δεν ήξεραν πώς να επεξεργαστούν την `<script>` ετικέτα. Δεδομένου ότι τα προγράμματα περιήγησης που κυκλοφόρησαν τα τελευταία 15 χρόνια δεν έχουν αυτό το πρόβλημα, αυτό το είδος σχολίου μπορεί να σας βοηθήσει να προσδιορίσετε τον πολύ παλιό κώδικα.

Εμφάνιση ειδοποίησης

Δημιουργήστε μια σελίδα που εμφανίζει ένα μήνυμα "Είμαι η JavaScript!".

Εξωτερικά αρχεία.

Εάν έχουμε πολύ κώδικα JavaScript, μπορούμε να τον βάλουμε σε ξεχωριστό αρχείο.

Τα αρχεία σεναρίων επισυνάπτονται σε HTML με το `src` χαρακτηριστικό:

```
<script src="/path/to/script.js"></script>
```

Εδώ, `/path/to/script.js`είναι μια απόλυτη διαδρομή προς το σενάριο από τη ρίζα του ιστότοπου. Κάποιος μπορεί επίσης να παρέχει μια σχετική διαδρομή από την τρέχουσα σελίδα. Για παράδειγμα, `src="script.js"`θα σήμαινε ένα αρχείο `"script.js"`στον τρέχοντα φάκελο.

Μπορούμε επίσης να δώσουμε μια πλήρη διεύθυνση URL. Για παράδειγμα:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js"></script>
```

Για να επισυνάψετε πολλά σενάρια, χρησιμοποιήστε πολλές ετικέτες:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
...
```

Εμφάνιση ειδοποίησης με εξωτερικό σενάριο

Ο κώδικας HTML:

```
<!DOCTYPE html>
<html>

<body>

  <script src="alert.js"></script>

</body>

</html>
```

Σχόλια

Με την πάροδο του χρόνου, τα προγράμματα γίνονται όλο και πιο περίπλοκα. Είναι απαραίτητο να προσθέσετε *σχόλια* που περιγράφουν τι κάνει ο κώδικας και γιατί.

Τα σχόλια μπορούν να τοποθετηθούν σε οποιοδήποτε μέρος ενός σεναρίου Δεν επηρεάζουν την εκτέλεση του, επειδή ο μεταγλωττιστής τα αγνοεί απλά.

Τα σχόλια μιας γραμμής ξεκινούν με δύο χαρακτήρες κάθετων //.

Το υπόλοιπο της γραμμής είναι ένα σχόλιο. Μπορεί να καταλάβει μια πλήρη γραμμή ή να ακολουθήσει μια δήλωση.

Οπως εδώ:

```
// This comment occupies a line of its own
alert('Hello');

alert('World'); // This comment follows the statement
```

Τα πολλαπλά σχόλια ξεκινούν με κάθετο και αστερίσκο /*και τελειώνουν με αστερίσκο και κάθετο */.

Σαν αυτό:

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
```

```
alert('World');
```

Το περιεχόμενο των σχολίων αγνοείται, οπότε αν βάλουμε κώδικα μέσα /* ... */, δεν θα εκτελεστεί.

Μερικές φορές μπορεί να είναι βολικό να απενεργοποιήσετε προσωρινά ένα μέρος του κώδικα:

Μεταβλητές

Τις περισσότερες φορές, μια εφαρμογή JavaScript πρέπει να λειτουργεί με πληροφορίες. Ακολουθούν δύο παραδείγματα:

1. Ένα ηλεκτρονικό κατάστημα - οι πληροφορίες μπορεί να περιλαμβάνουν εμπορεύματα που πωλούνται και καλάθι αγορών.
2. Μια εφαρμογή συνομιλίας - οι πληροφορίες μπορεί να περιλαμβάνουν χρήστες, μηνύματα και πολλά άλλα.

Οι μεταβλητές χρησιμοποιούνται για την αποθήκευση αυτών των πληροφοριών.

Μια μεταβλητή

Μια μεταβλητή είναι ένα "όνομα αποθήκευσης" για δεδομένα. Μπορούμε να χρησιμοποιήσουμε μεταβλητές για να αποθηκεύσουμε καλούδια, επισκέπτες και άλλα δεδομένα.

Για να δημιουργήσετε μια μεταβλητή σε JavaScript, χρησιμοποιήστε τη `let` λέξη-κλειδί.

Η παρακάτω δήλωση δημιουργεί (με άλλα λόγια: *δηλώνει*) μια μεταβλητή με το όνομα "μήνυμα":

```
let message;
```

Τώρα, μπορούμε να βάλουμε ορισμένα δεδομένα σε αυτό χρησιμοποιώντας τον χειριστή ανάθεσης `=`:

```
let message;

message = 'Hello'; // store the string
```

Η συμβολοσειρά αποθηκεύεται τώρα στην περιοχή μνήμης που σχετίζεται με τη μεταβλητή. Μπορούμε να το αποκτήσουμε χρησιμοποιώντας το όνομα της μεταβλητής:

```
let message;
message = 'Hello!';

alert(message); // shows the variable content
```

Για να είμαστε συνοπτικοί, μπορούμε να συνδυάσουμε τη μεταβλητή δήλωση και την εκχώρηση σε μία γραμμή:

```
let message = 'Hello!'; // define the variable and assign the value

alert(message); // Hello!
```

Μπορούμε επίσης να δηλώσουμε πολλές μεταβλητές σε μία γραμμή:

```
let user = 'John', age = 25, message = 'Hello';
```

Αυτό μπορεί να φαίνεται μικρότερο, αλλά δεν το προτείνουμε. Για καλύτερη αναγνωσιμότητα, χρησιμοποιήστε μία γραμμή ανά μεταβλητή.

Η παραλλαγή πολλαπλών γραμμών είναι λίγο μεγαλύτερη, αλλά πιο ευανάγνωστη:

```
let user = 'John';
let age = 25;
let message = 'Hello';
```

Μερικά άτομα ορίζουν επίσης πολλές μεταβλητές σε αυτό το στυλ πολλαπλών γραμμών:

```
let user = 'John',
  age = 25,
  message = 'Hello';
```

... Ή ακόμα και στο στυλ «κόμμα-πρώτο»:

```
let user = 'John'
  , age = 25
  , message = 'Hello';
```

Τεχνικά, όλες αυτές οι παραλλαγές κάνουν το ίδιο πράγμα. Είναι λοιπόν θέμα προσωπικής αισθητικής.

`var` αντί `let`

Σε παλαιότερα σενάρια, μπορείτε επίσης να βρείτε μια άλλη λέξη-κλειδί: `var` αντί `let`:

```
var message = 'Hello';
```

Η `var` λέξη-κλειδί είναι *σχεδόν* η ίδια με `let`. Δηλώνει επίσης μια μεταβλητή, αλλά με έναν ελαφρώς διαφορετικό,

Υπάρχουν λεπτές διαφορές μεταξύ `let` και `var`, αλλά δεν μας έχουν σημασία ακόμα. Θα τα καλύψουμε αναλυτικά σε επόμενα μαθήματα .

Μια πραγματική αναλογία

Μπορούμε εύκολα να κατανοήσουμε την έννοια μιας «μεταβλητής» αν τη φανταζόμαστε ως «κουτί» δεδομένων, με ένα αυτοκόλλητο με μοναδικό όνομα.

Για παράδειγμα, η μεταβλητή `message` μπορεί να φανταστεί ως ένα κουτί `"message"`με την τιμή `"Hello!"`σε αυτήν:

Μπορούμε να βάλουμε οποιαδήποτε αξία στο κουτί.

Μπορούμε επίσης να το αλλάξουμε όσες φορές θέλουμε:

```
let message;

message = 'Hello!';

message = 'World!'; // value changed

alert(message);
```

Όταν αλλάξει η τιμή, τα παλιά δεδομένα καταργούνται από τη μεταβλητή:

Μπορούμε επίσης να δηλώσουμε δύο μεταβλητές και να αντιγράψουμε δεδομένα από τη μία στην άλλη.

```
let hello = 'Hello world!';

let message;

// copy 'Hello world' from hello into message
message = hello;

// now two variables hold the same data
alert(hello); // Hello world!
alert(message); // Hello world!
```

Η διπλή δήλωση προκαλεί σφάλμα

Μια μεταβλητή πρέπει να δηλώνεται μόνο μία φορά.

Η επαναλαμβανόμενη δήλωση της ίδιας μεταβλητής είναι σφάλμα:

```
let message = "This";

// repeated 'let' leads to an error
let message = "That"; // SyntaxError: 'message' has already been declared
```

Επομένως, πρέπει να δηλώσουμε μια μεταβλητή μία φορά και στη συνέχεια να την αναφέρουμε χωρίς `let`.

Μεταβλητή ονομασία

Υπάρχουν δύο περιορισμοί στα ονόματα μεταβλητών στο JavaScript:

1. Το όνομα πρέπει να περιέχει μόνο γράμματα, ψηφία ή τα σύμβολα `$` και `_`.
2. Ο πρώτος χαρακτήρας δεν πρέπει να είναι ψηφίο.

Παραδείγματα έγκυρων ονομάτων:

```
let userName;
let test123;
```

Όταν το όνομα περιέχει πολλές λέξεις, το camelCase χρησιμοποιείται συνήθως. Αυτό είναι: οι λέξεις πάτε ένα μετά το άλλο, κάθε λέξη εκτός από την πρώτη εκκίνηση με ένα κεφαλαίο γράμμα: `myVeryLongName`.

Αυτό που είναι ενδιαφέρον - το σύμβολο του δολαρίου `'$'`και η υπογράμμιση `'_'`μπορούν επίσης να χρησιμοποιηθούν σε ονόματα. Είναι κανονικά σύμβολα, όπως τα γράμματα, χωρίς ιδιαίτερο νόημα.

Αυτά τα ονόματα είναι έγκυρα:

```
let $ = 1; // declared a variable with the name "$"
let _ = 2; // and now a variable with the name "_"

alert($ + _); // 3
```

Παραδείγματα λανθασμένων ονομάτων μεταβλητών:

```
let 1a; // cannot start with a digit

let my-name; // hyphens '-' aren't allowed in the name
```

Η υπόθεση έχει σημασία

Οι μεταβλητές ονομάζονται `apple` και `AppLE` είναι δύο διαφορετικές μεταβλητές.

Επιτρέπονται μη λατινικά γράμματα, αλλά δεν συνιστώνται

Είναι δυνατόν να χρησιμοποιήσετε οποιαδήποτε γλώσσα, συμπεριλαμβανομένων κυριλλικών γραμμάτων ή ακόμη και ιερογλυφικών, όπως αυτό:

```
let имя = '...';
let 我 = '...';
```

Από τεχνική άποψη, δεν υπάρχει σφάλμα εδώ. Τέτοια ονόματα επιτρέπονται, αλλά υπάρχει διεθνής σύμβαση για τη χρήση αγγλικών σε μεταβλητά ονόματα. Ακόμα κι αν γράφουμε ένα μικρό σενάριο, μπορεί να έχει μεγάλη διάρκεια ζωής. Άνθρωποι από άλλες χώρες μπορεί να χρειαστεί να το διαβάσουν κάποια στιγμή.

Δεσμευμένα ονόματα

Υπάρχει μια λίστα δεσμευμένων λέξεων , οι οποίες δεν μπορούν να χρησιμοποιηθούν ως μεταβλητά ονόματα, επειδή χρησιμοποιούνται από την ίδια τη γλώσσα.

Για παράδειγμα: `let`, `class`, `return`, και `function` είναι δεσμευμένες.

Ο παρακάτω κώδικας δίνει σφάλμα σύνταξης:

```
let let = 5; // can't name a variable "let", error!
let return = 5; // also can't name it "return", error!
```

Μια εργασία χωρίς `use strict`

Κανονικά, πρέπει να ορίσουμε μια μεταβλητή πριν τη χρησιμοποιήσουμε. Αλλά στα παλιά χρόνια, ήταν τεχνικά δυνατό να δημιουργηθεί μια μεταβλητή με μια απλή εκχώρηση της τιμής χωρίς χρήση `let`. Αυτό εξακολουθεί να λειτουργεί αν δεν βάλουμε `use strict` τα σενάρια μας για να διατηρήσουμε τη συμβατότητα με τα παλιά σενάρια.

```
// note: no "use strict" in this example

num = 5; // the variable "num" is created if it didn't exist

alert(num); // 5
```

Αυτή είναι μια κακή πρακτική και θα προκαλούσε σφάλμα σε αυστηρή λειτουργία:

```
"use strict";

num = 5; // error: num is not defined
```

Σταθερές

Για να δηλώσετε μια σταθερή (αμετάβλητη) μεταβλητή, χρησιμοποιήστε `const` αντί `let`:

```
const myBirthday = '18.04.1982';
```

Οι μεταβλητές που δηλώνονται χρησιμοποιώντας `const`ονομάζονται "σταθερές". Δεν μπορούν να εκχωρηθούν εκ νέου. Μια απόπειρα να προκαλέσει σφάλμα:

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

Όταν ένας προγραμματιστής είναι βέβαιος ότι μια μεταβλητή δεν θα αλλάξει ποτέ, μπορούν να την δηλώσουν με `const`εγγύηση και να κοινοποιήσουν σαφώς αυτό το γεγονός σε όλους.

Κεφαλαίες σταθερές

Υπάρχει μια ευρέως διαδεδομένη πρακτική να χρησιμοποιείτε σταθερές ως ψευδώνυμα για δύσκολες στη μνήμη τιμές που είναι γνωστές πριν από την εκτέλεση.

Αυτές οι σταθερές ονομάζονται με κεφαλαία γράμματα και κάτω παύλες.

Για παράδειγμα, ας φτιάξουμε σταθερές για χρώματα σε λεγόμενη μορφή "web" (δεκαεξαδική):

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...when we need to pick a color
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Οφέλη:

- `COLOR_ORANGE`είναι πολύ πιο εύκολο να το θυμάστε `"#FF7F00"`.
- Είναι πολύ πιο εύκολο να πληκτρολογήσετε λάθος `"#FF7F00"`από `COLOR_ORANGE`.
- Κατά την ανάγνωση του κώδικα, `COLOR_ORANGE`είναι πολύ πιο νόημα από `#FF7F00`.

Πότε πρέπει να χρησιμοποιούμε κεφαλαία για μια σταθερά και πότε πρέπει να το ονομάσουμε κανονικά; Ας το ξεκαθαρίσουμε.

Το να είσαι «σταθερός» σημαίνει ότι η τιμή μιας μεταβλητής δεν αλλάζει ποτέ Υπάρχουν όμως σταθερές που είναι γνωστές πριν από την εκτέλεση (όπως μια δεκαεξαδική τιμή για το κόκκινο) και υπάρχουν σταθερές που *υπολογίζονται* σε χρόνο εκτέλεσης, κατά την εκτέλεση, αλλά δεν αλλάζουν μετά την αρχική τους εκχώρηση.

Για παράδειγμα:

```
const pageLoadTime = /* time taken by a webpage to load */;
```

Η τιμή του `pageLoadTime`δεν είναι γνωστή πριν από τη φόρτωση της σελίδας, επομένως ονομάζεται κανονικά. Αλλά εξακολουθεί να είναι σταθερή επειδή δεν αλλάζει μετά την ανάθεση.

Με άλλα λόγια, οι σταθερές με όνομα κεφαλαίου χρησιμοποιούνται μόνο ως ψευδώνυμα για τις τιμές "hard-coded".

Ονομάστε τα πράγματα σωστά

Μιλώντας για μεταβλητές, υπάρχει ένα ακόμη εξαιρετικά σημαντικό πράγμα.

Ένα όνομα μεταβλητής πρέπει να έχει καθαρό, προφανές νόημα, που περιγράφει τα δεδομένα που αποθηκεύει.

Η μεταβλητή ονομασία είναι μία από τις πιο σημαντικές και πολύπλοκες δεξιότητες στον προγραμματισμό. Μια γρήγορη ματιά στα ονόματα μεταβλητών μπορεί να αποκαλύψει ποιος κωδικός γράφτηκε από έναν αρχάριο έναντι ενός έμπειρου προγραμματιστή.

Σε ένα πραγματικό έργο, ο περισσότερος χρόνος αφιερώνεται τροποποιώντας και επεκτείνοντας μια υπάρχουσα βάση κώδικα αντί να γράφουμε κάτι εντελώς ξεχωριστό από το μηδέν. Όταν επιστρέφουμε σε κάποιον κώδικα αφού κάνουμε κάτι άλλο για λίγο, είναι πολύ πιο εύκολο να βρεις πληροφορίες που έχουν καλή σήμανση. Ή, με άλλα λόγια, όταν οι μεταβλητές έχουν καλά ονόματα.

Αφιερώστε χρόνο για να σκεφτείτε το σωστό όνομα για μια μεταβλητή προτού την δηλώσετε. Κάτι τέτοιο θα σας αποπληρώσει πολύ.

Μερικοί κανόνες που πρέπει να ακολουθήσετε είναι:

- Χρησιμοποιήστε ονόματα αναγνώσιμα από τον άνθρωπο όπως `userName` ή `shoppingCart`.
- Μείνετε μακριά από συντομογραφίες ή μικρή ονόματα όπως `a`, `b`, `c`, εκτός αν ξέρετε πραγματικά τι κάνετε.
- Κάντε ονόματα με μέγιστο περιγραφικό και συνοπτικό τρόπο. Παραδείγματα κακών ονομάτων είναι `data` και `value`. Τέτοια ονόματα δεν λένε τίποτα. Είναι εντάξει να τα χρησιμοποιείτε μόνο εάν το περιεχόμενο του κώδικα το καθιστά εξαιρετικά προφανές σε ποια δεδομένα ή αξία αναφέρεται η μεταβλητή.

Επαναχρησιμοποίηση ή δημιουργία;

Και η τελευταία νότα. Υπάρχουν μερικοί τεμπέληδες προγραμματιστές που, αντί να δηλώνουν νέες μεταβλητές, τείνουν να επαναχρησιμοποιούν τις υπάρχουσες.

Ως αποτέλεσμα, οι μεταβλητές τους είναι σαν κουτιά στα οποία οι άνθρωποι ρίχνουν διαφορετικά πράγματα χωρίς να αλλάζουν τα αυτοκόλλητά τους. Τι υπάρχει μέσα στο κουτί τώρα; Ποιός ξέρει? Πρέπει να έρθουμε πιο κοντά και να ελέγξουμε.

Τέτοιοι προγραμματιστές εξοικονομούν λίγο στη μεταβλητή δήλωση, αλλά χάνουν δέκα φορές περισσότερο από τον εντοπισμό σφαλμάτων.

Μια επιπλέον μεταβλητή είναι καλή, όχι κακή.

Οι σύγχρονοι ελαχιστοποιητές JavaScript και τα προγράμματα περιήγησης βελτιστοποιούν τον κώδικα αρκετά καλά, οπότε δεν δημιουργεί προβλήματα απόδοσης. Η χρήση διαφορετικών μεταβλητών για διαφορετικές τιμές μπορεί ακόμη και να βοηθήσει τη μηχανή να βελτιστοποιήσει τον κώδικά σας.

Περίληψη

Μπορούμε να δηλώνουμε τις μεταβλητές για την αποθήκευση δεδομένων με τη χρήση των `var`, `let`ή `const` λέξεις-κλειδιά.

- `let` - είναι μια σύγχρονη μεταβλητή δήλωση.
- `var`- είναι μια δήλωση μεταβλητής παλιού τύπου.
- `const`- είναι σαν `let`, αλλά η τιμή της μεταβλητής δεν μπορεί να αλλάξει.

Οι μεταβλητές πρέπει να ονομάζονται με τρόπο που να μας επιτρέπει να κατανοήσουμε εύκολα τι υπάρχει μέσα τους.

Εργασία με μεταβλητές

1. Δηλώστε δύο μεταβλητές: `admin` και `name`.
2. Εκχωρήστε την τιμή `"John"` στην `name`.
3. Αντιγράψτε την τιμή από την `name` στην `admin`.
4. Δείξτε την τιμή της `admin` χρησιμοποιώντας την εντολή `alert` (πρέπει να εξάγετε "John").

Τύποι δεδομένων

Μια τιμή στην JavaScript είναι πάντα συγκεκριμένου τύπου. Για παράδειγμα, μια συμβολοσειρά ή έναν αριθμό.

Υπάρχουν οκτώ βασικοί τύποι δεδομένων σε JavaScript. Εδώ, θα τα καλύψουμε γενικά και στα επόμενα κεφάλαια θα μιλήσουμε για καθένα από αυτά λεπτομερώς.

Μπορούμε να βάλουμε οποιοδήποτε τύπο σε μια μεταβλητή. Για παράδειγμα, μια μεταβλητή μπορεί μια στιγμή να είναι συμβολοσειρά και, στη συνέχεια, να αποθηκεύσει έναν αριθμό:

```
// no error
let message = "hello";
message = 123456;
```

Αριθμός

```
let n = 123;
n = 12.345;
```

Ο τύπος *αριθμού* αντιπροσωπεύει αριθμούς ακέραιου και κινητού σημείου.

Υπάρχουν πολλές λειτουργίες για αριθμούς, π.χ. πολλαπλασιασμός `*`, διαίρεση `/`, προσθήκη `+`, αφαίρεση `-`και ούτω καθεξής.

Εκτός από την τακτική τους αριθμούς, υπάρχουν τα λεγόμενα «ειδικά αριθμητικές τιμές», που ανήκουν επίσης σε αυτόν τον τύπο δεδομένων: `Infinity`, `-Infinity` και `NaN`.

- `Infinity` αντιπροσωπεύει το μαθηματικό άπειρο ∞. Είναι μια ειδική τιμή που είναι μεγαλύτερη από οποιονδήποτε αριθμό.

Μπορούμε να το πάρουμε ως αποτέλεσμα της διαίρεσης με μηδέν:

```
alert( 1 / 0 ); // Infinity
```

Ή απλώς αναφέρετέ το απευθείας:

```
alert( Infinity ); // Infinity
```

- `NaN` αντιπροσωπεύει ένα υπολογιστικό σφάλμα. Είναι αποτέλεσμα λανθασμένης ή απροσδιόριστης μαθηματικής λειτουργίας, για παράδειγμα:

```
alert( "not a number" / 2 ); // NaN, such division is erroneous
alert( "not a number" / 2 + 5 ); // NaN
```

Οι μαθηματικές πράξεις είναι ασφαλείς

Το να κάνεις μαθηματικά είναι «ασφαλές» σε JavaScript. Μπορούμε να κάνουμε οτιδήποτε: διαιρέστε με μηδέν, αντιμετωπίζουμε μη αριθμητικές συμβολοσειρές ως αριθμούς κ.λπ.

Το σενάριο δεν θα σταματήσει ποτέ με ένα μοιραίο σφάλμα ("die"). Στη χειρότερη περίπτωση, θα έχουμε `NaN` το αποτέλεσμα.

Οι ειδικές αριθμητικές τιμές ανήκουν τυπικά στον τύπο "αριθμός".

BigInt

Στο JavaScript, ο τύπος "αριθμός" δεν μπορεί να αντιπροσωπεύει ακέραιες τιμές μεγαλύτερες από (δηλαδή) ή μικρότερες από τις αρνητικές. Είναι ένας τεχνικός περιορισμός που προκαλείται από την εσωτερική τους εκπροσώπηση.

$(2^{53}-1)9007199254740991-(2^{53}-1)$

Για τους περισσότερους σκοπούς αυτό είναι αρκετά, αλλά μερικές φορές χρειαζόμαστε πολύ μεγάλους αριθμούς, π.χ. για κρυπτογραφία ή χρονικές σφραγίδες ακριβείας μικροδευτερολέπτων.

`BigInt` ο τύπος προστέθηκε πρόσφατα στη γλώσσα για να αντιπροσωπεύει ακέραιους αριθμούς αυθαίρετου μήκους.

Δημιουργείται μια `BigInt` τιμή προσαρτώντας `n` στο τέλος ενός ακέραιου αριθμού:

```
// the "n" at the end means it's a BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

Ζητήματα συμβατότητας

Αυτή τη στιγμή, `BigInt`υποστηρίζεται στο Firefox / Chrome / Edge / Safari, αλλά όχι στο IE.

Σειρά

Μια συμβολοσειρά σε JavaScript πρέπει να περιβάλλεται από εισαγωγικά.

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

Στο JavaScript, υπάρχουν 3 τύποι εισαγωγικών.

1. Διπλά εισαγωγικά: `"Hello"`.
2. Μονά εισαγωγικά: `'Hello'`.
3. Βαρεία: `` `Hello` ``.(κυριολεκτικά)

Τα διπλά και μεμονωμένα εισαγωγικά είναι «απλά» εισαγωγικά. Δεν υπάρχει σχεδόν καμία διαφορά μεταξύ τους στο JavaScript.

Τα backticks είναι "εκτεταμένες λειτουργίες" εισαγωγικά. Μας επιτρέπουν να ενσωματώσουμε μεταβλητές και εκφράσεις σε μια συμβολοσειρά τυλίγοντας τις `${…}`, για παράδειγμα:

```
let name = "John";

// embed a variable
alert( `Hello, ${name}!` ); // Hello, John!

// embed an expression
alert( `the result is ${1 + 2}` ); // the result is 3
```

Η έκφραση μέσα `${…}`αξιολογείται και το αποτέλεσμα γίνεται μέρος της συμβολοσειράς. Μπορούμε να βάλουμε εκεί: μια μεταβλητή όπως `name` ή μια αριθμητική έκφραση όπως `1 + 2` ή κάτι πιο περίπλοκο.

Λάβετε υπόψη ότι αυτό μπορεί να γίνει μόνο με backticks

Δεν υπάρχει τύπος *χαρακτήρα* .

Σε ορισμένες γλώσσες, υπάρχει ένας ειδικός τύπος «χαρακτήρα» για έναν μόνο χαρακτήρα. Για παράδειγμα, στη γλώσσα C και στην Java ονομάζεται "char".

Στο JavaScript, δεν υπάρχει τέτοιος τύπος. Υπάρχει μόνο ένα είδος: `string`. Μια συμβολοσειρά μπορεί να αποτελείται από μηδενικούς χαρακτήρες (να είναι κενός), έναν χαρακτήρα ή πολλούς από αυτούς.

Boolean (λογικός τύπος)

Ο δυαδικός τύπος έχει μόνο δύο τιμές: `true` και `false`.

Αυτός ο τύπος χρησιμοποιείται συνήθως για την αποθήκευση τιμών ναι / όχι: `true` σημαίνει "ναι, σωστός" και `false` σημαίνει "όχι, λανθασμένος".

Για παράδειγμα:

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
```

Οι δυαδικές τιμές έρχονται επίσης ως αποτέλεσμα συγκρίσεων:

```
let isGreater = 4 > 1;

alert( isGreater ); // true (the comparison result is "yes")
```

Θα καλύψουμε βαθύτερα τα booleans στο κεφάλαιο Λογικοί τελεστές .

Η τιμή "null"

Η ειδική `null` τιμή δεν ανήκει σε κανέναν από τους τύπους που περιγράφονται παραπάνω.

Διαμορφώνει έναν ξεχωριστό τύπο που περιέχει μόνο την `null` τιμή:

```
let age = null;
```

Στο JavaScript, `null` δεν είναι «αναφορά σε ένα υπάρχον αντικείμενο» ή «μηδενικός δείκτης» όπως σε ορισμένες άλλες γλώσσες.

Είναι απλώς μια ειδική τιμή που αντιπροσωπεύει «τίποτα», «άδειο» ή «άγνωστη τιμή».

Η "απροσδιόριστη" τιμή

Η ειδική τιμή `undefined` ξεχωρίζει επίσης. Κάνει ένα δικό του είδος, όπως `null`.

Η έννοια του `undefined` είναι "η τιμή δεν έχει εκχωρηθεί".

Εάν μια μεταβλητή δηλώνεται, αλλά δεν έχει εκχωρηθεί, τότε η τιμή της είναι `undefined`:

```
let age;

alert(age); // shows "undefined"
```

Τεχνικά, είναι δυνατό να εκχωρηθεί ρητά `undefined` σε μια μεταβλητή:

```
let age = 100;

// change the value to undefined
age = undefined;

alert(age); // "undefined"
```

... Αλλά δεν συνιστούμε να το κάνετε αυτό. Κανονικά, κάποιος χρησιμοποιεί `null`για να αντιστοιχίσει μια "κενή" ή "άγνωστη" τιμή σε μια μεταβλητή, ενώ η `undefined` διατηρείται ως προεπιλεγμένη αρχική τιμή για πράγματα που δεν έχουν ανατεθεί.

Αντικείμενα και σύμβολα

Ο `object`τύπος είναι ειδικός.

Όλοι οι άλλοι τύποι ονομάζονται «πρωτόγονοι» επειδή οι τιμές τους μπορούν να περιέχουν μόνο ένα πράγμα (είτε πρόκειται για μια συμβολοσειρά ή έναν αριθμό ή οτιδήποτε άλλο). Αντίθετα, τα αντικείμενα χρησιμοποιούνται για την αποθήκευση συλλογών δεδομένων και πιο περίπλοκων οντοτήτων.

Όντας τόσο σημαντικά, τα αντικείμενα αξίζουν μια ειδική μεταχείριση. Θα τα εξετάσουμε αργότερα αφού μάθουμε περισσότερα για τους πρωταρχικούς τύπους δεδομένων

Ο `symbol` τύπος χρησιμοποιείται για τη δημιουργία μοναδικών αναγνωριστικών για αντικείμενα. Πρέπει να το αναφέρουμε εδώ για λόγους πληρότητας, αλλά επίσης να αναβάλουμε τις λεπτομέρειες μέχρι να γνωρίζουμε αντικείμενα.

Ο τύπος χειριστή typeof

Ο `typeof` τελεστής επιστρέφει τον τύπο δεδομένων της μεταβλητής. Είναι χρήσιμο όταν θέλουμε να επεξεργαστούμε διαφορετικές τιμές διαφορετικών τύπων ή απλώς να κάνουμε έναν γρήγορο έλεγχο.

Υποστηρίζει δύο μορφές σύνταξης:

1. Ως διαχειριστής: `typeof x`.
2. Ως συνάρτηση: `typeof(x)`.

Με άλλα λόγια, λειτουργεί με παρενθέσεις ή χωρίς αυτές. Το αποτέλεσμα είναι το ίδιο.

Η κλήση για `typeof x`επιστροφή μιας συμβολοσειράς με το όνομα τύπου:

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)
```

Οι τρεις τελευταίες γραμμές μπορεί να χρειάζονται πρόσθετη εξήγηση:

1. `Math` είναι ένα ενσωματωμένο αντικείμενο που παρέχει μαθηματικές λειτουργίες
2. Το αποτέλεσμα `typeof null` είναι `"object"`. Αυτό είναι ένα επίσημα αναγνωρισμένο σφάλμα στη `typeof` συμπεριφορά, που προέρχεται από τις πρώτες μέρες της JavaScript και διατηρείται για συμβατότητα. Σίγουρα, `null` δεν είναι αντικείμενο. Είναι μια ειδική τιμή με ξεχωριστό τύπο.

3. Το αποτέλεσμα `typeof alert` είναι `"function"`, γιατί η `alert`είναι μια συνάρτηση. Θα μελετήσουμε τις συναρτήσεις αργότερα όπου θα δούμε επίσης ότι δεν υπάρχει ειδικός τύπος «λειτουργίας» στο JavaScript. Οι συναρτήσεις ανήκουν στον τύπο αντικειμένου.

Περίληψη

Υπάρχουν 8 βασικοί τύποι δεδομένων σε JavaScript.

- `Number` για αριθμούς οποιουδήποτε είδους: ακέραιος ή κυμαινόμενο σημείο, οι ακέραιοι αριθμοί περιορίζονται από .$\pm(2^{53}-1)$
- `bigint` είναι για ακέραιους αριθμούς αυθαίρετου μήκους.

- `String` για συμβολοσειρές. Μια συμβολοσειρά μπορεί να έχει μηδενικούς ή περισσότερους χαρακτήρες, δεν υπάρχει ξεχωριστός τύπος ενός χαρακτήρα.
- `Boolean` για `true`/`false`.
- `Null` για άγνωστες τιμές - έναν αυτόνομο τύπο που έχει μία μόνο τιμή `null`.
- `Undefined` για μη εκχωρημένες τιμές - έναν αυτόνομο τύπο που έχει μία μόνο τιμή `undefined`.
- `object` για πιο περίπλοκες δομές δεδομένων.
- `symbol` για μοναδικά αναγνωριστικά.

  Ο `typeof` χειριστής μας επιτρέπει να δούμε ποιος τύπος αποθηκεύεται σε μια μεταβλητή.
- Δύο μορφές: `typeof x` ή `typeof(x)`.
- Επιστρέφει μια συμβολοσειρά με το όνομα του τύπου, όπως `"string"`.
- Για `null` επιστροφές `"object"`- πρόκειται για σφάλμα στη γλώσσα, δεν είναι στην πραγματικότητα αντικείμενο.

Συμβολοσειρά εισαγωγικά

Ποια είναι η έξοδος του σεναρίου;

```
let name = "Vangelis";

alert( `hello ${1}` ); // ?

alert( `hello ${"name"}` ); // ?

alert( `hello ${name}` ); // ?
```

Αλληλεπίδραση χρήστη: ειδοποίηση, προτροπή, επιβεβαίωση

Καθώς θα χρησιμοποιείτε το πρόγραμμα περιήγησης, όπως demo του περιβάλλοντός μας, ας δούμε μερικές λειτουργίες να αλληλεπιδρούν με τον χρήστη: `alert`, `prompt` και `confirm`.

συναγερμός

Αυτό το έχουμε ήδη δει. Εμφανίζει ένα μήνυμα και περιμένει ο χρήστης να πατήσει "OK".

Για παράδειγμα:

```
alert("Hello");
```

Το μίνι-παράθυρο με το μήνυμα ονομάζεται *modal παράθυρο* . Η λέξη "modal" σημαίνει ότι ο επισκέπτης δεν μπορεί να αλληλεπιδράσει με την υπόλοιπη σελίδα, να πατήσει άλλα κουμπιά κ.λπ., έως ότου έχει αντιμετωπίσει το παράθυρο. Σε αυτήν την περίπτωση - έως ότου πατήσουν "OK".

προτροπή

Η συνάρτηση `prompt` δέχεται δύο ορίσματα:

```
result = prompt(title, [default]);
```

Εμφανίζει ένα παράθυρο τρόπου με ένα μήνυμα κειμένου, ένα πεδίο εισαγωγής για τον επισκέπτη και τα κουμπιά OK / Cancel.

`title`

Το κείμενο που θα δείξει στον επισκέπτη.

`default`

Μια προαιρετική δεύτερη παράμετρος, η αρχική τιμή για το πεδίο εισαγωγής.

Οι αγκύλες τετραγώνου σε σύνταξη `[...]`

Οι αγκύλες τετραγώνου γύρω `default` από τη σύνταξη παραπάνω υποδηλώνουν ότι η παράμετρος είναι προαιρετική, δεν απαιτείται.

Ο επισκέπτης μπορεί να πληκτρολογήσει κάτι στο πεδίο εισαγωγής προτροπής και να πατήσει OK. Στη συνέχεια, λαμβάνουμε αυτό το κείμενο στο `result`. Ή μπορούν να ακυρώσουν την είσοδο πατώντας Άκυρο ή πατώντας το `Esc` πλήκτρο, τότε παίρνουμε `null` ως `result`.

Η κλήση για `prompt` επιστροφή του κειμένου από το πεδίο εισαγωγής ή `null` εάν η εισαγωγή ακυρώθηκε.

Για παράδειγμα:

```
let age = prompt('How old are you?', 100);

alert(`You are ${age} years old!`); // You are 100 years old!
```

Επιβεβαίωση

Η σύνταξη:

```
result = confirm(question);
```

Η συνάρτηση `confirm` δείχνει ένα παράθυρο τρόπου με ένα `question` και δύο κουμπιά: OK και Ακύρωση.

Το αποτέλεσμα είναι `true` εάν πατηθεί OK και `false` εάν πατηθεί “Cancel”

Για παράδειγμα:

```
let isBoss = confirm("Are you the student?");

alert( isBoss ); // true if OK is pressed
```

Περίληψη

Καλύψαμε 3 λειτουργίες ειδικά για το πρόγραμμα περιήγησης για αλληλεπίδραση με τους επισκέπτες:

`alert`

δείχνει ένα μήνυμα.

`prompt`

εμφανίζει ένα μήνυμα που ζητά από το χρήστη να εισαγάγει κείμενο. Επιστρέφει το κείμενο ή, εάν Ακύρωση κουμπί ή `Esc` κάνετε κλικ, `null`.

`confirm`

εμφανίζει ένα μήνυμα και περιμένει ο χρήστης να πατήσει "OK" ή "Ακύρωση". Επιστρέφει `true` για OK και `false` για Cancel / `Esc`.

Όλες αυτές οι μέθοδοι είναι τροπικές: διακόπτουν την εκτέλεση σεναρίου και δεν επιτρέπουν στον επισκέπτη να αλληλεπιδράσει με την υπόλοιπη σελίδα έως ότου το παράθυρο έχει απορριφθεί.

Υπάρχουν δύο περιορισμοί που μοιράζονται όλες οι παραπάνω μέθοδοι:

1. Η ακριβής θέση του παραθύρου καθορίζεται από τον browser Συνήθως, είναι στο κέντρο.
2. Η ακριβής εμφάνιση του παραθύρου εξαρτάται επίσης από το πρόγραμμα περιήγησης. Δεν μπορούμε να το τροποποιήσουμε.

Μια απλή σελίδα

Δημιουργήστε μια ιστοσελίδα που ζητά ένα όνομα και να το παρουσιάζει.

Εκτελέστε την επίδειξη

λύση

Κωδικός JavaScript:

```
let name = prompt("What is your name?", "");
alert(name);
```

Η πλήρης σελίδα:

```
<!DOCTYPE html>
<html>
<body>

  <script>
    'use strict';

    let name = prompt("What is your name?", "");
    alert(name);
  </script>

</body>
</html>
```

Τύπος Μετατροπές

Τις περισσότερες φορές, οι χειριστές και οι λειτουργίες μετατρέπουν αυτόματα τις τιμές που τους έχουν δοθεί στον σωστό τύπο.

Για παράδειγμα, `alert` μετατρέπει αυτόματα οποιαδήποτε τιμή σε μια συμβολοσειρά για να την εμφανίσει. Οι μαθηματικές πράξεις μετατρέπουν τιμές σε αριθμούς.

Υπάρχουν επίσης περιπτώσεις όπου πρέπει να μετατρέψουμε ρητά μια τιμή στον αναμενόμενο τύπο.

Μετατροπή συμβολοσειράς

Η μετατροπή συμβολοσειράς συμβαίνει όταν χρειαζόμαστε τη μορφή συμβολοσειράς μιας τιμής.

Για παράδειγμα, το `alert(value)` κάνει για να δείξει την τιμή.

Μπορούμε επίσης να καλέσουμε τη `String(value)` συνάρτηση για να μετατρέψουμε μια τιμή σε μια συμβολοσειρά:

```
let value = true;
alert(typeof value); // boolean

value = String(value); // now value is a string "true"
alert(typeof value); // string
```

Η μετατροπή συμβολοσειρών είναι ως επί το πλείστον προφανής. Ένα `false` γίνεται `"false"`, `null` γίνεται `"null"`, κ.λπ.

Αριθμητική μετατροπή

Η αριθμητική μετατροπή συμβαίνει αυτόματα σε μαθηματικές συναρτήσεις και εκφράσεις.

Για παράδειγμα, όταν `/` εφαρμόζεται διαίρεση σε μη αριθμούς:

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

Μπορούμε να χρησιμοποιήσουμε τη `Number(value)` συνάρτηση για να μετατρέψουμε ρητά έναν `value` σε αριθμό:

```
let str = "123";
alert(typeof str); // string

let num = Number(str); // becomes a number 123
```

```
alert(typeof num); // number
```
Απαιτείται ρητή μετατροπή όταν διαβάζουμε μια τιμή από μια πηγή που βασίζεται σε συμβολοσειρά, όπως μια φόρμα κειμένου, αλλά αναμένουμε να εισαχθεί ένας αριθμός.

Εάν η συμβολοσειρά δεν είναι έγκυρος αριθμός, το αποτέλεσμα μιας τέτοιας μετατροπής είναι `NaN`. Για παράδειγμα:

```
let age = Number("string instead of a number");

alert(age); // NaN, conversion failed
```
Αριθμητικοί κανόνες μετατροπής:

αξία	Γίνεται...
`undefined`	`NaN`
`null`	`0`
`true and false`	`1` και `0`
`string`	Τα κενά διαστήματα από την αρχή και το τέλος καταργούνται. Εάν η υπόλοιπη συμβολοσειρά είναι κενή, το αποτέλεσμα είναι `0`. Διαφορετικά, ο αριθμός «διαβάζεται» από τη συμβολοσειρά. Ένα σφάλμα δίνει `NaN`.

Παραδείγματα:

```
alert( Number("   123   ") ); // 123
alert( Number("123z") );      // NaN (error reading a number at "z")
alert( Number(true) );        // 1
alert( Number(false) );       // 0
```
Σημειώστε ότι `null`και `undefined`συμπεριφέρεστε διαφορετικά εδώ: `null`γίνεται μηδέν ενώ `undefined`γίνεται `NaN`.

Οι περισσότεροι μαθηματικοί τελεστές εκτελούν επίσης τέτοια μετατροπή, θα το δούμε στο επόμενο κεφάλαιο.

Boolean μετατροπή

Η Boolean μετατροπή είναι η απλούστερη.

Συμβαίνει σε λογικές λειτουργίες (αργότερα θα συναντήσουμε δοκιμές συνθηκών και άλλα παρόμοια πράγματα), αλλά μπορεί επίσης να εκτελεστεί ρητά με μια κλήση προς `Boolean(value)`.

Για παράδειγμα:

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false

alert( Boolean("hello") ); // true
alert( Boolean("") ); // false
```
Σημειώστε: η συμβολοσειρά με μηδέν `"0"` είναι `true`

Περίληψη

Οι τρεις πιο ευρέως χρησιμοποιούμενες μετατροπές τύπου είναι η συμβολοσειρά, ο αριθμός και το boolean.

`String Conversion`- Εμφανίζεται όταν βγάζουμε κάτι. Μπορεί να εκτελεστεί με `String(value)`. Η μετατροπή σε συμβολοσειρά είναι συνήθως προφανής για πρωτόγονες τιμές.

`Numeric Conversion`- Εμφανίζεται σε μαθηματικές πράξεις. Μπορεί να εκτελεστεί με `Number(value)`.

Η μετατροπή ακολουθεί τους κανόνες:

αξία	Γίνεται...
`undefined`	`NaN`

αξία	Γίνεται...
`null`	`0`
`true / false`	`1 / 0`
`string`	Η συμβολοσειρά διαβάζεται «ως έχει», τα κενά και από τις δύο πλευρές αγνοούνται. Μια κενή συμβολοσειρά γίνεται `0`. Ένα σφάλμα δίνει `NaN`.

`Boolean Conversion`- Εμφανίζεται σε λογικές λειτουργίες. Μπορεί να εκτελεστεί με `Boolean(value)`.

Ακολουθεί τους κανόνες:

αξία	Γίνεται...
`0`, `null`, `undefined`, `NaN`, `""`	`false`
οποιαδήποτε άλλη τιμή	`true`

Βασικοί τελεστές, μαθηματικά

Γνωρίζουμε πολλούς χειριστές από το σχολείο. Είναι πράγματα όπως η προσθήκη `+`, ο πολλαπλασιασμός `*`, η αφαίρεση `-` και ούτω καθεξής.

Σε αυτό το κεφάλαιο, θα ξεκινήσουμε με απλούς τελεστές και, στη συνέχεια, θα επικεντρωθούμε σε συγκεκριμένες πτυχές JavaScript, που δεν καλύπτονται από τη βασική αριθμητική.

Μαθηματικά

Υποστηρίζονται οι ακόλουθες μαθηματικές λειτουργίες:

- Προσθήκη `+`,
- Αφαίρεση `-`,
- Πολλαπλασιασμός `*`,
- Διαίρεση `/`,
- Υπόλοιπο `%`,
- Ύψωση σε δύναμη `**`.

Οι πρώτες τέσσερις είναι απλή, ενώ για `%` και `**`χρειάζονται λίγα λόγια γι 'αυτούς.

Υπόλοιπο %

Ο υπόλοιπος χειριστής `%`, παρά την εμφάνισή του, δεν σχετίζεται με τα ποσοστά.

Το αποτέλεσμα `a % b`είναι το υπόλοιπο της ακέραιας διαίρεσης `a` από το `b`.

Για παράδειγμα:

```
alert( 5 % 2 ); // 1, a remainder of 5 divided by 2
alert( 8 % 3 ); // 2, a remainder of 8 divided by 3
```

Ύψωση σε δύναμη **

Ο τελεστής ύψωσης σε δύναμη `a ** b` πολλαπλασιάζεται το `a` , `b` φορές.

Για παράδειγμα:

```
alert( 2 ** 2 ); // 4  (2 multiplied by itself 2 times)
alert( 2 ** 3 ); // 8  (2 * 2 * 2, 3 times)
alert( 2 ** 4 ); // 16 (2 * 2 * 2 * 2, 4 times)
```

Μαθηματικά, η ύψωση σε δύναμη ορίζεται και για μη ακέραιους αριθμούς. Για παράδειγμα, μια τετραγωνική ρίζα είναι εκθετική με `1/2`:

```
alert( 4 ** (1/2) ); // 2 (power of 1/2 is the same as a square root)
alert( 8 ** (1/3) ); // 2 (power of 1/3 is the same as a cubic root)
```

Συνδυασμός συμβολοσειράς με δυαδικό +

Ας συναντήσουμε τις δυνατότητες των χειριστών JavaScript που είναι πέρα από τη βασικήαριθμητική.

Συνήθως, ο χειριστής συν `+`αθροίζει τους αριθμούς.

Αλλά, εάν το δυαδικό `+`εφαρμοστεί σε συμβολοσειρές, τα συγχωνεύει (συνδυάζει):

```
let s = "my" + "string";
alert(s); // mystring
```
Σημειώστε ότι εάν κάποιος από τους τελεστές είναι συμβολοσειρά, τότε ο άλλος μετατρέπεται και σε συμβολοσειρά.

Για παράδειγμα:

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```
Βλέπετε, δεν έχει σημασία αν ο πρώτος τελεστής είναι μια συμβολοσειρά ή ο δεύτερος.

Ακολουθεί ένα πιο περίπλοκο παράδειγμα:

```
alert(2 + 2 + '1' ); // "41" and not "221"
```
Εδώ, οι χειριστές δουλεύουν το ένα μετά το άλλο. Ο πρώτος `+`αθροίζει δύο αριθμούς, οπότε επιστρέφει `4`και στη συνέχεια ο επόμενος `+` προσθέτει τη συμβολοσειρά `1`σε αυτό, έτσι είναι σαν `4 + '1' = 41`.

Το δυαδικό `+` είναι ο μόνος τελεστής που υποστηρίζει συμβολοσειρές με τέτοιο τρόπο. Άλλοι αριθμητικοί τελεστές λειτουργούν μόνο με αριθμούς και μετατρέπουν πάντα τους τελεστές τους σε αριθμούς.

Εδώ είναι το demo για αφαίρεση και διαίρεση:

```
alert( 6 - '2' ); // 4, converts '2' to a number
alert( '6' / '2' ); // 3, converts both operands to numbers
```

Προτεραιότητα χειριστή

Εάν μια παράσταση έχει περισσότερους από έναν τελεστές, η εντολή εκτέλεσης καθορίζεται από την *προτεραιότητά τους* ή, με άλλα λόγια, από την προεπιλεγμένη σειρά προτεραιότητας των τελεστών.

Από το σχολείο, όλοι γνωρίζουμε ότι ο πολλαπλασιασμός στην έκφραση `1 + 2 * 2` πρέπει να υπολογίζεται πριν από την προσθήκη. Αυτό είναι ακριβώς το θέμα προτεραιότητας. Ο πολλαπλασιασμός λέγεται ότι έχει *μεγαλύτερη προτεραιότητα* από την προσθήκη.

Οι παρενθέσεις παρακάμπτουν κάθε προτεραιότητα, οπότε αν δεν είμαστε ικανοποιημένοι με την προεπιλεγμένη σειρά, μπορούμε να τις χρησιμοποιήσουμε για να την αλλάξουμε. Για παράδειγμα, γράψτε `(1 + 2) * 2`.

Υπάρχουν πολλοί τελεστές σε JavaScript. Κάθε χειριστής έχει έναν αντίστοιχο αριθμό προτεραιότητας. Εκείνος με τον μεγαλύτερο αριθμό εκτελείται πρώτα. Εάν η προτεραιότητα είναι η ίδια, η εντολή εκτέλεσης είναι από αριστερά προς τα δεξιά.

Αύξηση / μείωση

Η αύξηση ή μείωση ενός αριθμού κατά έναν είναι από τις πιο κοινές αριθμητικές πράξεις.

Έτσι, υπάρχουν ειδικοί χειριστές για αυτό:

- **Η αύξηση** `++` αυξάνει μια μεταβλητή κατά 1:

```
let counter = 2;
counter++;        // works the same as counter = counter + 1, but is shorter
alert( counter ); // 3
```

- **Η μείωση** `--` μειώνει μια μεταβλητή κατά 1:

```
let counter = 2;
counter--;        // works the same as counter = counter - 1, but is shorter
alert( counter ); // 1
```

Σπουδαίος:

Η αύξηση / μείωση μπορεί να εφαρμοστεί μόνο σε μεταβλητές. Προσπαθώντας να το χρησιμοποιήσετε σε μια τιμή όπως `5++` θα δώσει ένα σφάλμα.

Οι χειριστές `++`και `--`μπορούν να τοποθετηθούν είτε πριν είτε μετά από μια μεταβλητή.

- Όταν ο χειριστής πηγαίνει μετά τη μεταβλητή, είναι σε «μορφή postfix»: `counter++`.
- Η «μορφή πρόθεμα» είναι όταν ο χειριστής πηγαίνει πριν από τη μεταβλητή: `++counter`.

Και οι δύο αυτές δηλώσεις κάνουν το ίδιο πράγμα: αύξηση `counter`κατά `1`.

Υπάρχει διαφορά; Ναι, αλλά μπορούμε να το δούμε μόνο εάν χρησιμοποιήσουμε την επιστρεφόμενη τιμή του `++/--`.

Ας διευκρινίσουμε. Όπως γνωρίζουμε, όλοι οι χειριστές επιστρέφουν μια τιμή. Η αύξηση / μείωση δεν αποτελεί εξαίρεση. Η φόρμα προθέματος επιστρέφει τη νέα τιμή ενώ η φόρμα μετά την επιδιόρθωση επιστρέφει την παλιά τιμή (πριν από την αύξηση / μείωση).

Για να δείτε τη διαφορά, ακολουθεί ένα παράδειγμα:

```
let counter = 1;
let a = ++counter; // (*)

alert(a); // 2
```

```
let counter = 1;
let a = counter++; // (*) changed ++counter to counter++

alert(a); // 1
```

Συγκρίσεις

Γνωρίζουμε πολλούς τελεστές σύγκρισης από μαθηματικά.

Στο JavaScript γράφονται ως εξής:

- Μεγαλύτερο / μικρότερο από: `a > b`, `a < b`.
- Μεγαλύτερη / μικρότερο ή ίσον: `a >= b`, `a <= b`.
- Ισούται: `a == b` παρακαλώ σημειώστε ότι το σύμβολο διπλής ισότητας `==`σημαίνει έλεγχος ισότητας, ενώ το `a = b` σημαίνει ανάθεση.
- Όχι ίσο. Στα μαθηματικά η σημειογραφία είναι `≠`, αλλά σε JavaScript γράφεται ως `a != b`.

Το Boolean είναι το αποτέλεσμα

Όλοι οι τελεστές σύγκρισης επιστρέφουν μια δυαδική τιμή:

- `true` - σημαίνει "ναι", "σωστός" ή "η αλήθεια".
- `false` - σημαίνει "όχι", "λάθος" ή "όχι η αλήθεια".

Για παράδειγμα:

```
alert( 2 > 1 );  // true (correct)
alert( 2 == 1 ); // false (wrong)
alert( 2 != 1 ); // true (correct)
```

Ένα αποτέλεσμα σύγκρισης μπορεί να αντιστοιχιστεί σε μια μεταβλητή, όπως και οποιαδήποτε τιμή:

```
let result = 5 > 4; // assign the result of the comparison
alert( result ); // true
```

Σύγκριση συμβολοσειρών

Για να δείτε αν μια συμβολοσειρά είναι μεγαλύτερη από μια άλλη, το JavaScript χρησιμοποιεί τη λεγόμενη σειρά "λεξικό" ή "λεξικογραφική".

Για παράδειγμα:

```
alert( 'Z' > 'A' ); // true
alert( 'Glow' > 'Glee' ); // true
alert( 'Bee' > 'Be' ); // true
```
Ο αλγόριθμος για τη σύγκριση δύο συμβολοσειρών είναι απλός:

1. Συγκρίνετε τον πρώτο χαρακτήρα και των δύο χορδών.
2. Εάν ο πρώτος χαρακτήρας από την πρώτη συμβολοσειρά είναι μεγαλύτερος (ή μικρότερος) από τους άλλους, τότε η πρώτη συμβολοσειρά είναι μεγαλύτερη (ή μικρότερη) από τη δεύτερη. Τελειώσαμε.
3. Διαφορετικά, εάν οι πρώτοι χαρακτήρες και των δύο χορδών είναι οι ίδιοι, συγκρίνετε τους δεύτερους χαρακτήρες με τον ίδιο τρόπο.
4. Επαναλάβετε μέχρι το τέλος κάθε συμβολοσειράς.
5. Εάν και οι δύο χορδές τελειώνουν στο ίδιο μήκος, τότε είναι ίσες. Διαφορετικά, η μεγαλύτερη συμβολοσειρά είναι μεγαλύτερη.

Στο πρώτο παράδειγμα παραπάνω, η σύγκριση `'Z' > 'A'` φτάνει στο αποτέλεσμα στο πρώτο βήμα.

Σύγκριση διαφορετικών τύπων

Κατά τη σύγκριση τιμών διαφορετικών τύπων, το JavaScript μετατρέπει τις τιμές σε αριθμούς.

Για παράδειγμα:

```
alert( '2' > 1 ); // true, string '2' becomes a number 2
alert( '01' == 1 ); // true, string '01' becomes a number 1
```
Για δυαδικές τιμές, `true` γίνεται `1` και `false` γίνεται `0`.

Για παράδειγμα:

```
alert( true == 1 ); // true
alert( false == 0 ); // true
```
.

Αυστηρή ισότητα

Ένας τακτικός έλεγχος ισότητας `==`έχει πρόβλημα. Δεν μπορεί να διαφοροποιηθεί π.χ το `0` από το `false`:

```
alert( 0 == false ); // true
```
Το ίδιο συμβαίνει με μια κενή συμβολοσειρά:

```
alert( '' == false ); // true
```
Αυτό συμβαίνει επειδή οι τελεστές διαφορετικών τύπων μετατρέπονται σε αριθμούς από τον τελεστή ισότητας `==`. Μια κενή συμβολοσειρά, όπως `false`, γίνεται μηδέν.

Τι να κάνετε αν θα θέλαμε να διαφοροποιηθούν `0` από `false`;

Ένας αυστηρός χειριστής ισότητας `===`ελέγχει την ισότητα χωρίς μετατροπή τύπου.

Με άλλα λόγια, εάν `a` και `b` είναι διαφορετικών τύπων, τότε `a === b` επιστρέφει αμέσως `false` χωρίς προσπάθεια μετατροπής τους.

Ας το προσπαθήσουμε:

```
alert( 0 === false ); // false, because the types are different
```
Υπάρχει επίσης ένας «αυστηρός μη ισότητα» χειριστής `!==`ανάλογος με `!=`.

Σύγκριση με μηδενική και απροσδιόριστη

Για αυστηρό έλεγχο ισότητας `===`

Αυτές οι τιμές είναι διαφορετικές, επειδή καθεμία από αυτές είναι διαφορετικού τύπου.

```
alert( null === undefined ); // false
```
Για μη αυστηρό έλεγχο `==`

```
alert( null == undefined ); // true
```
Για μαθηματικά και άλλες συγκρίσεις `< > <= >=`