JavaScript Grammar

introduction

Variables

- Labels which refer to a changeable value.

- Example:

  total may be possess a value of 100.

Operators

- Actors which can be used to calculate or compare values.
- Example:

  Two values may be summed using the addition operator (+);

  ```
  total+tax
  ```

- Example:

  Two values may be compared using the greater-than operator (>);

  ```
  total>200
  ```

Expressions

- Any combination of variables, operators, and statements which evaluate to some result.
- In English parlance this might be termed a "sentence" or even a "phrase", in that grammatical elements are combined into a cogent meaning.
- Example:

```
total=100;
```

- Example:

```
if (total>100)
```

Statements

- As in English, a statement pulls all grammatical elements together into a full thought.
- JavaScript statements may take the form of conditionals, loops, or object manipulations.
- It is good form to separate statements by semicolons, although this is only mandatory if multiple statements reside on the same line.
- Example:

```
if (total>100)
{statements;}
else {statements;}
```

- Example:

```
while (clicks<10)
{statements;}
```

Objects

- Containing constructs which possess a set of values, each value reflected into an individual property of that object.

- Objects are a critical concept and feature of JavaScript.

- A single object may contain many properties, each property which acts like a variable reflecting a certain value.

- JavaScript can reference a large number of "built-in" objects which refer to characteristics of a Web document.

- For instance, the document object contains properties which reflect the background color of the current document, its title, and many more. For a fuller explanation of the built-in objects of JavaScript, see the section on "Document Object Model".

Functions and Methods

- A JavaScript function is quite similar to a "procedure" or "subroutine" in other programming languages.
- A function is a discrete set of statements which perform some action. It may accept incoming values (parameters), and it may return an outgoing value.
- A function is "called" from a JavaScript statement to perform its duty.
- A method is simply a function which is contained in an object.
- For instance, a function which closes the current window, named `close()`, is part of the `window` object; thus, `window.close()` is known as a method.

Variables

- Variables store and retrieve data, also known as "values".
- A variable can refer to a value which changes or is changed.
- Variables are referred to by name, although the name you give them must conform to certain rules.
- A JavaScript identifier, or name, must start with a letter or underscore ("_"); subsequent characters can also be digits (0-9).
- Because JavaScript is case sensitive, letters include the characters "A" through "Z" (uppercase) and the characters "a" through "z" (lowercase).

Scope

- If you want to create a local variable which only scopes within that function you must declare the new variable using the var statement:

Text

```
function newFunction()
{ var loop=1;
  total=0;
  ...additional statements...
}
```

- In the example above, the variable loop will be local to newFunction(), while total will be global to the entire page.

Types: Numbers

- **Integers** can be expressed in *decimal* (base 10), *hexadecimal* (base 16), and *octal* (base 8).
- A decimal integer literal consists of a sequence of digits without a leading 0 (zero).
- A leading 0 (zero) on an integer literal indicates it is in octal; a leading 0x (or 0X) indicates hexadecimal.
- Hexadecimal integers can include digits (0-9) and the letters a-f and A-F. Octal integers can include only the digits 0-7.
- A **floating-point** number can contain either a decimal point, an "e" (uppercase or lowercase), which is used to represent "ten to the power of" in scientific notation, or both.
- The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").
- A floating-point literal must have at least one digit and either a decimal point or "e" (or "E").

Types: Booleans

- The possible Boolean values are true and false.
- These are special values, and are not usable as 1 and 0.
- In a comparison, any expression that evaluates to 0 is taken to be false, and any statement that evaluates to a number other than 0 is taken to be true.

Other types

Strings

- "Hello World !"
  Strings are delineated by single or double quotation marks. (Use single quotes to type strings that contain quotation marks.)

Objects

- myObj = new Object();

Null

- Not the same as zero - no value at all. A *null* value is one that has no value and means nothing.

Undefined

- A value that is *undefined* is a value held by a variable after it has been created, but before a value has been assigned to it.

Dynamically typed

"5" + "10" => "510" (string concatenation)

5 + 10 => 15 (arithmetic sum)

Operators

+	Addition
-	Subtraction
*****	Multiplication
/	Division
%	Modulus: the remainder after division; e.g. 10 % 3 yields 1.
++	Unary increment: this operator only takes one operand. The operand's value is increased by 1. The value returned depends on whether the ++ operator is placed before or after the operand; e.g. *++x* will return the value of *x* following the increment whereas *x++* will return the value of *x* prior to the increment.
--	Unary decrement: this operator only takes one operand. The operand's value is decreased by 1. The value returned depends on whether the -- operator is placed before or after the operand; e.g. *--x* will return the value of *x* following the decrement whereas *x--* will return the value of *x* prior to the decrement.
-	Unary negation: returns the negation of operand.

Comparison

==	"Equal to" returns true if operands are equal.
!=	"Not equal to" returns true if operands are not equal.
>	"Greater than" returns true if left operand is greater than right operand.
>=	"Greater than or equal to" returns true if left operand is greater than or equal to right operand.
<	"Less than" returns true if left operand is less than right operand.
<=	"Less than or equal to" returns true if left operand is less than or equal to right operand.

&&	"And" returns true if both operands are true.
||	"Or" returns true if either operand is true.
!	"Not" returns true if the negation of the operand is true (e.g. the operand is false).

Booleans and Assignment

=	Assigns the value of the righthand operand to the variable on the left. Example: total=100; Example: total=(price+tax+shipping)
+= (also -=, *=, /=)	Adds the value of the righthand operand to the lefthand variable and stores the result in the lefthand variable. Example: total+=shipping (adds value of *shipping* to *total* and assigned result to *total*)
&= (also |=)	Assigns result of (lefthand operand && righthand operand) to lefthand operand.

Special operators

Conditional operator **(condition) ? trueVal : falseVal**	Assigns a specified value to a variable if a condition is true, otherwise assigns an alternate value if condition is false. Example: preferredPet = (cats > dogs) ? "felines" : "canines" If (cats>dogs), *preferredPet* will be assigned the string value "felines," otherwise it will be assigned "canines".
typeof *operand*	Returns the data type of *operand*. Example -- test a variable to determine if it contains a number: if (typeof total=="number") ...

support for **regular expressions**, which are defined patterns used to match character combinations appearing in string values

Statements

- **Block**

  A set of statements that is surrounded by braces is called a block. Blocks of statements are used, for example, in functions and conditionals.

- **Conditionals**

  Conditional statements direct program flow in specified directions depending upon the outcomes of specified conditions.

Conditionals

```
if (condition)
 { statements1; }
else
 { statements2; }
```

Conditionals

```
switch (expression){
    case label :
        statement;
        break;
    case label :
        statement;
        break;
    ...
    default : statement;
}
```

```
switch (favoritePet){
   case "dog" :
      statements;
      break;
   case "cat" :
      statements;
      break;
   case "iguana" :
      statements;
      break;
   default : statements;
}
```

Loops

```
for (initial-statement; test; increment)
 { statements; }
```

The initial-statement is executed first, and once only. Commonly, this statement is used to initialize a counter variable.
Then the test is applied and if it succeeds then the statements are executed.
The increment is applied to the counter variable and then the loop starts again.

For loop

```
for (i=0; i<10; i++)
 { statements; }
```

Do While

```
do
  { statements;}
while (condition)
```

- executes a block of statements repeatedly until a condition becomes false.
- Due to its structure, this loop necessarily executes the statement at least once

While

```
while (condition)
 { statements; }
```

- executes its statement block as long as the condition is true.
- The main difference between `while` and `do...while` is that a while loop may not execute the statements even once if the condition is initially false

break	Aborts execution of the loop, drops out of loop to the next statement following the loop.
continue	Aborts *this single* iteration of the loop, returns execution to the loop control, meaning the condition specified by the loop statement. Loop may execute again if condition is still true.

```
for (variable in object)
 { statements; }

var record = "Wine1<br><br>"
for (var prop in wine1)
 {record += prop + " = " + wine1[prop] + "<BR>"}
record += "<br>"
document.write(record)
```

- The sometimes confusing for...in statement is used to cycle through each property of an object or each element of an array.
- The idea is that you may want to execute a statement block which operates on every property or element

with

```
with (object)
 { statements; }
```

- The with statement serves as a sort of shorthand, allowing you to execute a series of statement who all assume a specified object as the reference.
- In other words, the object specified in the with statement is used as the default object whenever a property is encountered with no object specified

Comments

```
//A lonely ol' single line comment

/* A dense thicket of commentary, spanning
many captivating lines
of explanation and intrigue. */
```

- For **single line** comments, simply precede the line with two backslashes.
- For **multi-line** comment blocks, begin the comment with /* and close with */.

Functions

```
function Name(argument1,argument2,etc)
{ statements; }
```

- A function doesn't necessarily require arguments, in which case you need only write out the parenthesis; e.g. funcName().

- Generally it's best to define the functions for a page in the HEAD portion of a document.
- Since the HEAD is loaded first, this guarantees that functions are loaded before the user has a chance to do anything that might call a function.
- Alternately, some programmers place all of their functions into a separate file, and include them in a page using the SRC attribute of the SCRIPT tag.
- Either way, the key is to load the function definitions before any code is executed.

```
function boldblink(message)
{ document.write("<blink><strong>"+message
+"</strong></blink>"); }
```

Calling functions

```
clearPage();

boldblink("Chiamami cretino!");
```

- A function waits in the wings until it is called onto the stage.
- You call a function simply by specifying its name followed by a parenthetical list of arguments, if any:

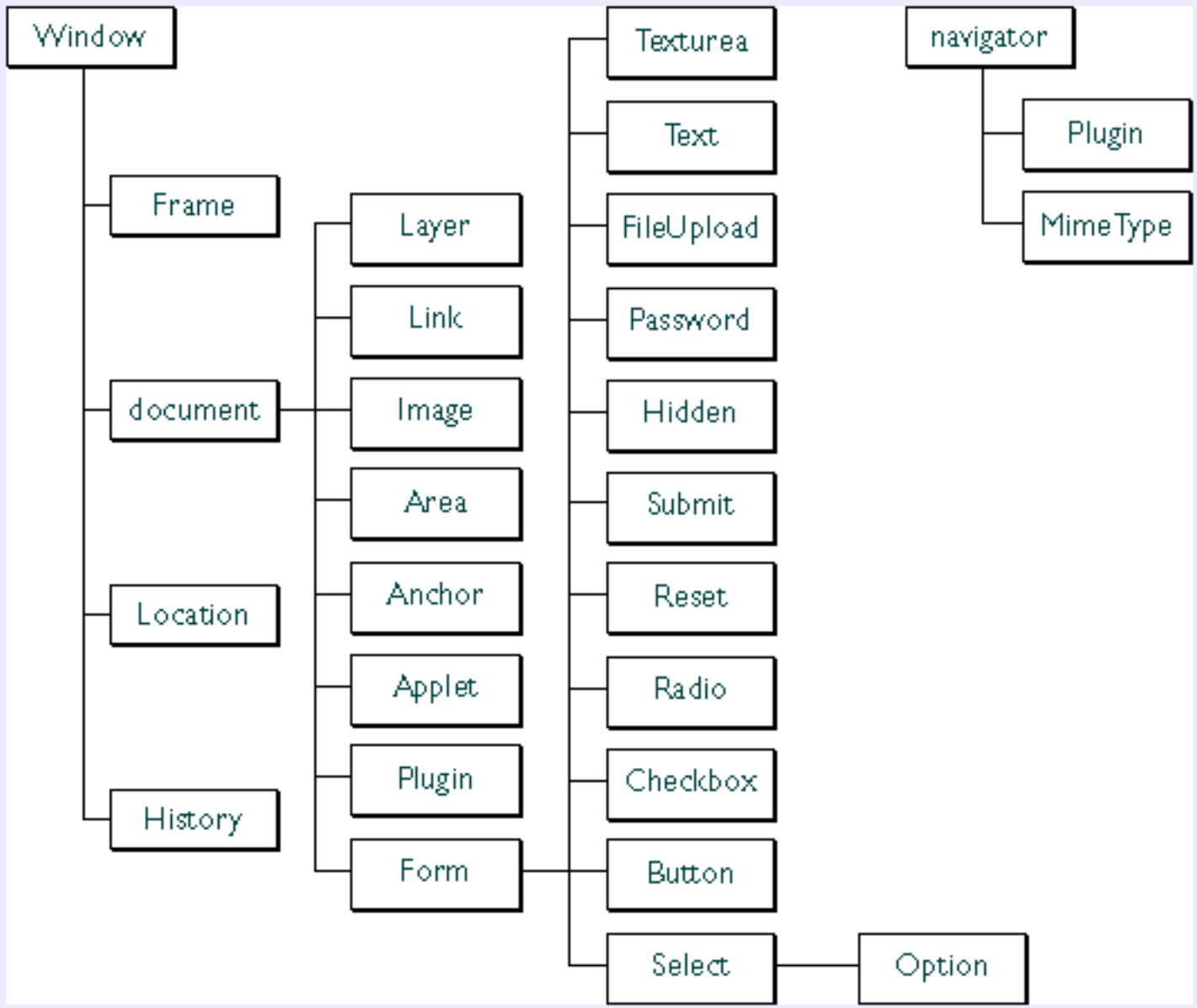
Objects

- An object is a "package" of data; **a collection of properties** (variables) **and methods** (functions) all classed under a single name.
- For example, imagine that there was an object named car.
- We could say that the car object possesses several properties: make, model, year, and color, for example.
- We might even say that car possesses some methods: go(), stop(), and reverse().
- Although car is obviously fictional, you can see that its properties and methods all relate to a common theme

Document Object Model

- Often referred to as the DOM, this object model is a hierarchy of all objects "built in" to JavaScript.
- Most of these objects are directly related to characteristics of the Web page or browser.
- The reason we qualify the term "built in" is because the DOM is technically separate from JavaScript itself.
- That is, the JavaScript language specification, standardized by the ECMA, does not actually specify the nature or specifics of the DOM.
- Consequently, Netscape and Microsoft have developed their own individual DOM's which are not entirely compatible.
- ECMA = European Computer Manufacturers Association, Geneva, Switzerland

DOM Model



Reprinted from Netscape's JavaScript Guide