DATA TYPES

Javascript data types

Javascript is a **loosely typed language**, meaning you do not have to declare the **data types** of variables explicitly. Javascript uses a **variant datatype** and performs conversions automatically as necessary.

JavaScript has six types of data, usually referred to as **primitive data types**. The main types are **strings, numbers, objects**, and **Booleans**. The other two are **null** and **undefined** .

For example, if you try to add a number to an item that consists of text (a string), the number is automatically converted to text. While this is often very convenient, it doesn't mean you can ignore data types altogether.

String Data Type

Strings are any text contained within single or double quotation marks. (Use single quotes to type strings that contain quotation marks.) A string is also an **object** in Javascript, but it is a special case, with special properties.

To illustrate one of the pitfalls of a loosely typed language, consider the following example which calculates the area and perimeter of a rectangle. In the example the four variables are all examples of strings:

```
function test()
{
   var myText =  "The perimeter is ";
   var Width  =  "50";
   var Height =  "20";

   // output the perimeter of a rectangle
   // e.g. 2* (Width plus Height)
   alert("The perimeter is " + 2*(Width + Height));
   Width = parseInt(Width);Height = parseInt(Height);
   alert("Wrong! the answer should be " + 2*(Width + Height));
}
```

To work out the area javascript correctly deduces that Width and Height are supposed to be numbers and multiples them together.
When working out the perimeter, the width and height are concatenated (as strings) instead of being added together (as numbers). e.g. 5020 rather than

70.
This value is then (correctly) multiplied by 2 to give 10040 which is obviously wrong and could be a serious problem if it goes unnoticed.

Number Data Type

Javascript supports both integer and floating-point numbers. Integers can be positive, 0, or negative; 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. These numbers follow the IEEE 754 standard for numerical representation. Last, there are certain number values that are special:

- **NaN** , or **N**ot **a N**umber
- Positive Infinity
- Negative Infinity
- Positive 0
- Negative 0

By default Integers are represented in base 10 (decimal), but may also represent numbers in base 16 (hexadecimal) and base 8 (octal). The computer of course only understands binary (or base 2) arithmetic.

Hexadecimal and octal numbers can be negative, but cannot be fractional. A number that begins with a single "0" (zero) and contains a decimal point is a decimal floating-point number; if a number that begins with "0x" or "00" contains a decimal point, anything to the right of the decimal point is ignored.

Hexadecimal ("hex") integers are specified by a leading "0x" (the "X" can be uppercase or lowercase) and can contain digits 0 through 9 and letters A through F (either uppercase or lowercase). The letter "e" is a permissible digit in hexadecimal notation and **does not signify an exponential number**. The letters A through F are used to represent, as single digits, the numbers that are 10 through 15 in base 10. That is, 0xF is equivalent to 15, and 0x10 is equivalent to 16.
You will come across hexadecimal numbers when assigning color values in HTML

Octal integers are specified by a leading "0", and can contain digits 0 through 7. If a number has a leading "0" but contains the digits "8" and/or "9", it is a decimal number. A number that would otherwise be an octal number but contains the letter "e" (or "E") generates an error.

Some example numbers:

```
    .0001, 0.0001, 1e-4, // 3 floating-point numbers, equivalent to
each other.
```

```
    3.45e2                // A floating-point number, equivalent to
345 decimal.
    42                    // An integer number.
    0377                  // An octal integer, equivalent to 255
decimal.
    0378                  // An integer, equivalent to 378 decimal.
    00.0001               // Octal numbers cannot have decimal
parts,
                          // this is equivalent to 0.
    0Xff                  // A hexadecimal integer, equivalent to
255 decimal.
    0x37CF                // A hexadecimal integer, equivalent to
14287 decimal.
    0x3e7                 // A hexadecimal integer, equivalent to
999 decimal.
                          // i.e 3 * 16 * 16 + E (decimal 14) * 16 +
7 = 999
    0x3.45e2              // Hexadecimal numbers cannot have decimal
parts,
                          // this is equivalent to 3.
```

Booleans

The possible Boolean values are **true** and **false.** These are special values, and are not usable as 1 and 0, however in a logical context, (automatic) type coercion evaluates 0, -0, **null, NaN, undefined** or the empty string ("") as false. Everything else evaluates as true.

> **Note** 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**. Thus the following expression evaluates to true.

```
    (false == 0) // Note the double equal sign for comparison
```

Undefined Data Type A value that is undefined is simply a value given to a variable after it has been created, but before a value has been assigned to it.

> **Note:** if you are familiar with Visual basic, then this is a significant difference. In Visual Basic for example when you declare an array as integer its values are automatically set to 0. In Javascript when you declare **any** variable its value is **undefined** until you **assign** a value to it**.**

Null Data Type

A **null** value is one that has no value and means nothing

Source : http://www.soslug.org/node/1722