ななちゃんのIT教室

原始人の背比べ？　の巻

by nara.yasuhiro@gmail.com

ななちゃんが、JavaScript のプリミティブと、
比較（等価演算子）について学ぶという お話

第 0.1 版　2017 年 6 月 19 日

フリー素材
http://freeillustration.net

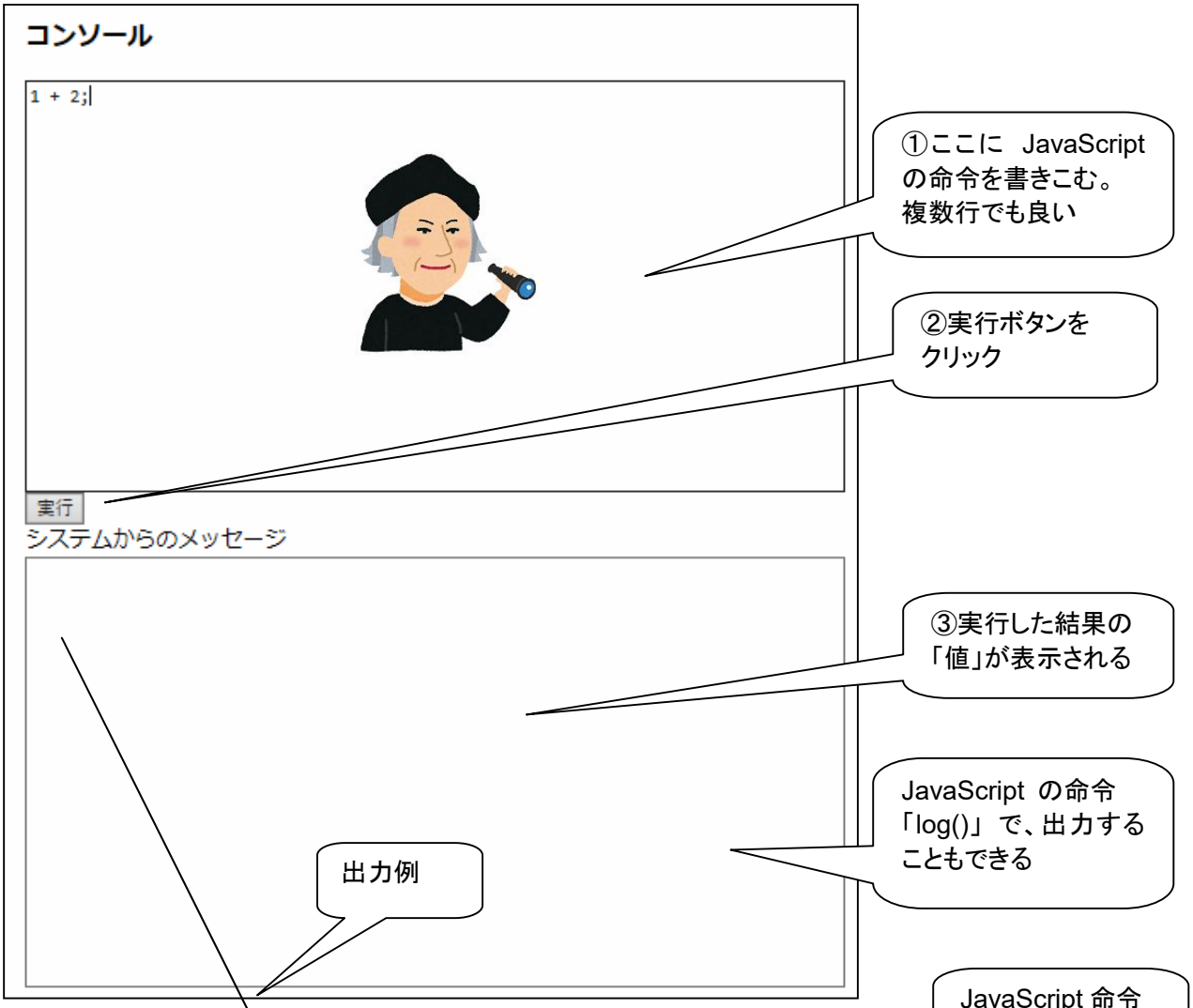
いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回　秘密道具：マイ・コンソール

なな：クリじい、「原始人の背比べ」の勉強をするんだけど、便利な秘密道具はない？

クリ：あるぞ、あるぞ。定番秘密道具の「マイ・コンソール」。他の巻を読んでない読者のために、説明しよう。




```
<= 1 + 2;
=> Number number 3
<= "1" + "2"
=> String string "12"
<= 1;2;
=> Number number 2
<= var x = 1;
=> Undefined undefined undefined
<= x
=> Number number 1
<= var x; x= 1;
=> Number number 1
```

JavaScript 命令「1+2」を入力した

```
<= 1 + 2;
=> Number number 3
```

実行結果の「値」は 3

注意：var x = 1; の値は「undefined」

実行結果の「型」は Number

本 教材ではこのように圧縮表示しています

```
1 + 2;          // Number number 3
"1" + "2"       // String string "12"
1;2;            // Number number 2
var x = 1;      // Undefined undefined undefined
x               // Number number 1
var x; x= 1;    // Number number 1
```

「型」の判定方法は 2 種類 r

「○；○」のように、複数の JavaScript 命令がある場合、一番右の命令の型、値だけ表示される

JavaScript 命令

実行結果の「型」と「値」

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>コンソール</title>
 </head>
 <body>
    <h3>コンソール</h3>
    <textarea rows="19" cols="80" id=pg autofocus>1 + 2;</textarea>
    <br><input type=button onClick=go() value="実行">
    <br>システムからのメッセージ
    <br><textarea rows="20" cols="80" id=log></textarea>
    <script>
var geval = eval;

var logp = document.getElementById("log");
var pgp  = document.getElementById("pg");
var logd;

function clog(s) { logp.value += s; }
function log(s) { logd += s; }
function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }
function isPrimitive(x) {
    return (typeof x)!="object";
}
function toLiteral(x) {
    if (typeIs(x)=="Number" && isNaN(x)) return "NaN";
    if (x === Infinity)                  return "Infinity";
    if ((typeIs(x)!="Symbol")&&(-x === Infinity)) return "-Infinity";
    if (typeIs(x)=="Set") return "Set("+JSON.stringify([...x])+")";
    if (typeIs(x)=="Map") return "Map("+JSON.stringify([...x])+")";
    return JSON.stringify(x);
}
function type(x) { return "" + (typeof x); }
function isInteger(n) { return n%1 === 0; }
function keys(obj) { return Object.keys(obj); }
function go() {
 logd = "";
 try {
            var v = geval(pgp.value);
            clog("<= " + pgp.value + "\n=> "
               + typeIs(v) + " " + type(v) + " " + toLiteral(v) + "\n");
            pgp.value = "";
            logp.scrollTop = logp.scrollHeight;
            pgp.focus();
 }
 catch(e) { clog("<= " + pgp.value + "\n=>! " + e + "\n");
                pgp.value = "";
                logp.scrollTop = logp.scrollHeight;
                pgp.focus();
 }
 if (logd != "") clog(logd + "\n");
}
   </script>
 </body>
</html>
```

第2回　プリミティブ(primitive、プリミティブ値、プリミティブデータ型)

なな：今回のお題の「プリミティブ」って何？

先生：primitive は、原始の、初期の、太古の、昔の、幼稚な、素朴な、根本の、基本のという意味の英語。JavaScript のデータは、基本的に、すべて、オブジェクトですが、その例外がプリミティブ。JavaScript には 6 個のプリミティブデータ型があります。文字列、数値、真偽値、null、undefined、そして、ECMAScript 2015 で追加された シンボル。

なな：今回は、どんなスーパーツールを使うの？

クリ：myConsole ！　入力した JavaScript の文（式）を評価して、その値の型と値自身を表示できるんじゃ。型は、「Object.prototype.toString.call(値).slice(8, -1))」と「typeof 値」のふたつの方法で表示する。片方だけでは区別がつかないものを区別することができる場合がある。

型 1　型 2　値

```
"abc"          // String string "abc"
1              // Number number 1
true           // Boolean boolean true
null           // Null object null
undefined      // Undefined undefined undefined
Symbol()       // Symbol symbol undefined
```

先生：数値プリミティブには、整数、小数点数、負数、といった一般的な数以外に、特別なものとして、Nan、Infinity があります。NaN は、Not a Number、Infinity は、無限大。

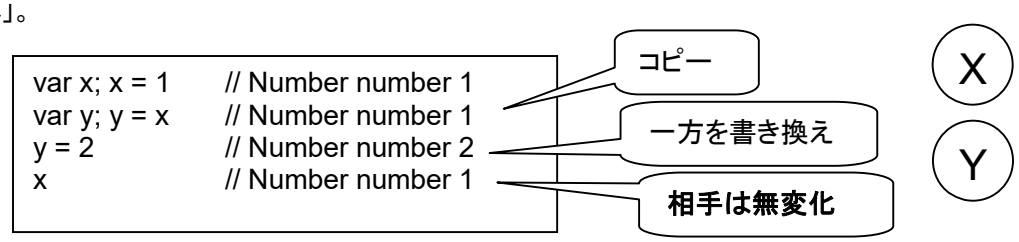
```
NaN                    // Number number NaN
Number("abc")          // Number number NaN
NaN == NaN             // Boolean boolean false

var nan1; nan1 = NaN   // Number number NaN
var nan2; nan2 = nan1  // Number number NaN
nan1 == nan2           // Boolean boolean false

(3/0)                  // Number number Infinity
Infinity - Infinity    // Number number NaN
Infinity + Infinity    // Number number Infinity
5 * Infinity           // Number number Infinity
0 * Infinity           // Number number NaN
```

要注意！

第3回　参照渡しと値渡し

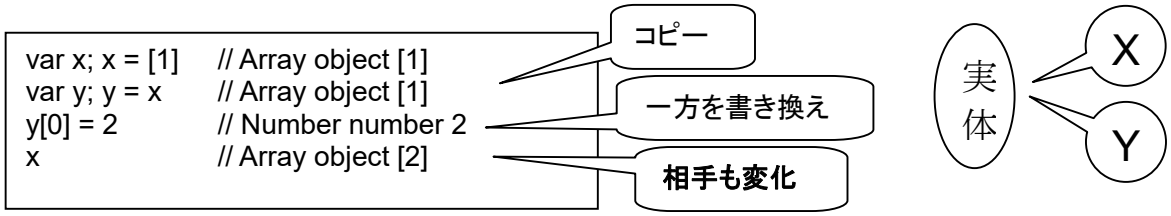
なな：参照渡しと値渡し？　何、それ。

先生：プリミティブ値は、変数に**データそのもの**として記憶します。これが「**値渡し**」。プリミティブ値以外の、オブジェクトは、コンピュータの記憶領域にデータを置き、変数には、データの、記憶装置上の**アドレス**を記憶します。これが「**参照渡し**」。

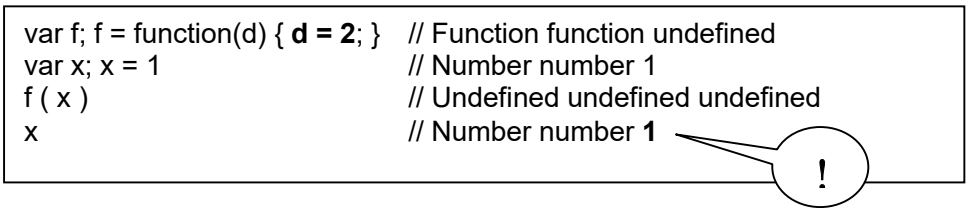


なな：それが、そんな影響を及ぼすの？

先生：**値渡し**の場合、y = x（代入）では、**データのコピー**が行われます。あとで、コピーのほうを書き換えても、オリジナルのほうはそのままです。

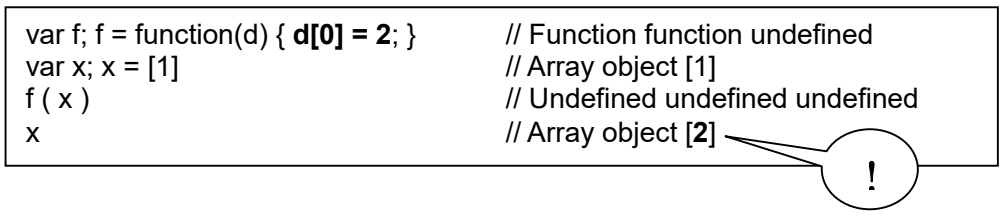


先生：配列（Array）は、プリミティブではないので、**参照渡し**。y = x は、**アドレスのコピー**になるので、**データはひとつ**で、x も y も、**同じデータを指し示します**。ですから、y を書き換えると、x も変化します。ポイントは、y = 2 ではなく、y[0] = 2 というところ。y のアドレスのところのデータを 2 に書き換えるということで、アドレスは変化しません。



なな：代入の場合ね。関数呼び出しの場合は？

先生：**値渡し**の場合、関数（メソッド）の**引数として渡すのはデータのコピー**。だから、メソッドの中でデータを書き換えても、呼び出し元でそれを読み出すことができません。



先生：**参照渡し**の場合、メソッドの**引数として渡すのはアドレス**。だから、メソッドの中でデータを書き換えて、呼び出し元でそれを読み出すことができます。読み出せるということは、逆に言えば、気づかないうちに、データが変化している（書き換えられている）ということにもなります。

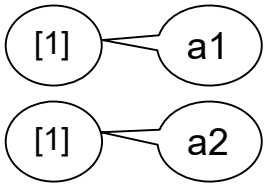
なな：データの比較はどうなるの？

```
var n1; n1 = 1          // Number number 1
var n2; n2 = 1          // Number number 1
n1 == n2                // Boolean boolean true
n1 === n2               // Boolean boolean true
```

先生：**数字**のように**値渡し**の場合、== や === による比較は、**値の比較**になります。↑

値は同じでも異なる実態

```
var a1; a1 = [ 1 ]      // Array object [1]
var a2; a2 = [ 1 ]      // Array object [1]
a1 == a2                // Boolean boolean false
a1 === a2               // Boolean boolean false
```



先生：この例（↑）では、1 という要素を持つ**配列**をふたつ作り、それぞれのアドレスを a1 と a2 に記憶します。== や === は、値の比較ではなく、**アドレスの比較**になります。配列はプリミティブではないので**参照渡し**になります。

```
var s1; s1 = "abc"      // String string "abc"
var s2; s2 = "abc"      // String string "abc"
s1 == s2                // Boolean boolean true
s1 === s2               // Boolean boolean true
```

先生：**文字列**は、文字の並びであり、配列に似ていますが、JavaScript では、どんなに長い文字列でも**値渡し**になります。== や === は、**文字列の内容が同じであるかどうかを比較**します。文字列はプリミティブです。

第4回　等価演算子 == と 厳密等価演算子 ===

なな：== と、=== の違いは？

先生：下記の例を見てください。

```
1 == "1"          // Boolean boolean true
1 === "1"         // Boolean boolean false
[1] == [1]        // Boolean boolean false
[1] === [1]       // Boolean boolean false
[1]+"" == 1       // Boolean boolean true
[1]+"" === 1      // Boolean boolean false
```

先生：**==** は、「**等価演算子**」と言います。数値と文字列を比較するとき、**文字列は数値に変換された上で比較**します。比較するふたつの**データの型が異なる場合**、一定のルールによって、片方の**データを変換**し、**型をそろえた上で**、「値」を比較します。比較するふたつの**データの型が同じ場合**、そのまま比較します。データが**プリミティブ型（値渡し）なら、データ（値）の比較**になります。データが**オブジェクト（参照渡し）なら、アドレスの比較**になります。

なな：じゃあ、=== のほうは？

先生：**===** は、「**厳密等価演算子**」と言います。**型を変換することなく**、ふたつの記憶内容を直接比較します。**型が異なれば、即、false になります。型が同じ場合**、データが**プリミティブ型（値渡し）なら、データ（値）の比較**になります。**データがオブジェクト（参照渡し）なら、アドレスの比較**になります。

なな：==（等価演算子）で、**true、false と数値の比較**は？

先生：true は 1 に、false は 0 に変換されます。

```
true == 1         // Boolean boolean true
false == 0        // Boolean boolean true
true == 3         // Boolean boolean false
false == 3        // Boolean boolean false
```

なな：誕生日は同じ（5 月 5 日）（ == ）でも、年齢は違う（1981 年と 1975 年）（ === ）とか、
名前なしの電車チケットは譲渡可（ == ）だけど、名前付きの航空券は譲渡不可（ === ）みたいな。

先生：ちょっと苦しいけど、まあ、そんなところかも。

なな：お店で買えばそれっきり（ ==、値渡し）だけど、通信販売だと買ったあと、ダイレクトメールの広告が届く（ ===、参照渡し）とか。

先生：まあ、そんなことかも。通信販売では「今後ダイレクトメール希望しない」をちゃんと指定しましょう。

第5回 if 文とプリミティブ

先生：それから、ちょっと飛躍するけど、if(...) の中のデータが論理値でなかった場合の扱いも説明しておきましょう。if の中身が x とした場合、「x == true」が成り立つなら実行、「x == false」が成り立つなら非実行、というふうに確認するのは不正確です。true→1、false→0 に変換された上で数値として比較されてしまう可能性があるからです。!!x（! は NOT）が true か false かを調べるのが正確です。if で、true 扱いは、「{}」、「中身のある文字列」、「0 以外の数値」、「配列」。false 扱いは、「""」、「0」、「undefined」、「null」です。

空文字列

なな：わざわざ、分かりにくいことをやらなくて良いのに。

先生：たとえば、棒グラフを描くために、数を星の数で表す場合、

```
var n = 3, s = "";
for (var i=0; i<n; i++) s += "*";
s;                          // String string "***"
```

を

```
var var n = 3; s = "";
while(n--) s += "*";
s;                          // String string "***"
```

と書くと、パチパチ！ ということもあるわけね。NaN は曲者で、

```
NaN == NaN                  // Boolean boolean false
NaN === NaN                 // Boolean boolean false

NaN == false                // Boolean boolean false
NaN == true                 // Boolean boolean false
!NaN                        // Boolean boolean true
var x=0; if (NaN) x=1; x    / Number number 0
var x=0; if (!NaN) x=1; x   // Number number 1
```

という性質があります。たとえば、<input> から入力した文字列型の数字(str)を、data = Number(str)で数字に変換しようとするとき、文字列が "abc" のような、数字でなかった場合、NaN が返されます。そのまま計算を続けるとエラーになるので、if(data == NaN) alert("error") としたくなります。でも、NaN == NaN が true にならないので、思ったような動作はしません。この問題は多発するので、**isNaN()** という関数が用意されています。

```
isNaN(NaN)    // Boolean boolean true
isNaN(12)     // Boolean boolean false
```

のような動作をします。前の例では、if (isNaN(data)) alert("error") とすればOKです。null や、undefined は、

```
null == null                // Boolean boolean true
null === null               // Boolean boolean true
undefined == undefined      // Boolean boolean true
undefined === undefined     // Boolean boolean true
```

となるので、if(data == null) { ... } や、if(data === null) { ... } のように使えます。蛇足ですが、

```
null == undefined           // Boolean boolean true
null === undefined          // Boolean boolean false

!null                       // Boolean boolean true
```

という扱いなので、data が、通常は数字で、問題がある時に null になる場合は、

```
If (!data) alert( "error" );
```

という書き方ができますが、data が 0 の場合も実行してしまうので要注意です。もっと正確には、null、undefined、0、空文字("")、false の場合に実行します。ですから、

```
If (data === null) alert( "error" );
```

のほうが安全です。

なな：そもそも、undefined なんて、何に使うの？

先生：こんな使い方があります。関数の引数が省略された場合、undefined になります。

```
function show(name) {
   if (name === undefined) name = '名無';
   return("こんにちは、" + name + "さん");
}
show();                    // String string "こんにちは、名無さん"
show('JavaScript');        // String string "こんにちは、JavaScript さん"
```

クリ：undefined を使わないくても書けるけどね。

```
function show(name) {
   if (arguments.length < 1) name = '名無';
   return("こんにちは、" + name + "さん");
}
```

ECMA Script 2015 なら、こういう書き方もある。

```
function show(name = '名無') {
   return("こんにちは、" + name + "さん");
}
```

なな：じゃあ、null は、何に使うの？

クリ：たとえば、String.match() メソッドなんかは、文字が見つからなかった時に null を返すぞ。

```
"abc".match(/b/)          // Array object ["b"]
"abc".match(/x/)          // Null object null
```

だから、「文字列が見つからなかった場合は」というのを、

```
if (str.match(/x/) == null) alert( "みつからなかった" )
```

のような使い方をするんじゃ。JavaScript 組み込みのメソッドの多くは、問題が発生した時に null を返してくるんじゃ。

先生：そうね。undefined は、データが得られなかった、null は、「問題あり」というデータが得られた、というところかしらねえ。

第6回　オブジェクトをプリミティブ値に変換するルール

なな：==（等価演算子）で、 プリミティブ値と、オブジェクトを比較する場合、 オブジェクトのほうは、 どうやってプリミティブ値に変換するの？まさか、プリミティブ値をオブジェクトに変換するはずはないよね。

先生：そうね。オブジェクトを、どうやってプリミティブ値に変換するかを説明しましょう。オブジェクトを、プリミティブ値に変換するには、オブジェクトのvalueOf() や toString() メソッドを使います。前者を優先します。そうした変換の試みが失敗したときには実行時にエラーになります。

```
new String("foo") == "foo"        // Boolean boolean true
[1,2] == "1,2"                    // Boolean boolean true
```

これらは、オブジェクトが文字列に変換されてから比較され、true になります（ == ）。

```
"foo" === new String("foo")       // Boolean boolean false
```

これは、変換しないで比較されるので false になります（ === ）。

```
(new String("foo")) == (new String("foo"))        //    Boolean boolean false
```

これは、オブジェクト同士の、アドレスの比較なので false になります。下記は、valueOf() が使われていることを見る、ちょっとした実験です。

クリ：じっけん、じっけん！

```
var s; s = new String("foo")                // String object "foo"
"foo" == s                                  // Boolean boolean true

s.valueOf = function() { return "boo"; }  // Function function undefined
"foo" == s                                  // Boolean boolean false
"boo" == s                                  // Boolean boolean true
```

valueOf メソッドをいたずらしたら、コンピュータが騙されてしまった。
Numberオブジェクトではない、一般のオブジェクトに ValueOf メソッドを設けると、あたかも Numberオブジェクトのような動作をするんじゃ。下記の例を見ちくれ。

```
var o; o = { valueOf: function(){ return 1; } } // Object object {}
o == 1                    // Boolean boolean true
o+""                      // String string "1"
o + o                     // Number number 2
```

上記の オブジェクト o は、まるで 数値 1 のようにふるまうんじゃ。

```
var o; o = { value:1,valueOf:function(){ return this.value; }} // Object object {"value":1}
o+""                      // String string "1"
var o2; o2 = o            // Object object {"value":1}
o2.value = 2              // Number number 2
o+""                      // String string "2"
```

上記の オブジェクトo も、まるで 数値のようにふるまうが、参照渡しなので、コピーしてから値を変えると、オリジナルのほうまで変化する。これは、あくまで、知的遊びじゃ。実際のプログラムで使うんじゃないぞ。

なな：はあ～い。

第7回　ラッパーオブジェクト

先生：null と undefined を除くすべてのプリミティブ値には、そのプリミティブ値を内包する等価のラッパーオブジェクトがあります。

なな：ラッパー？　パンパカパーンのラッパ？　ラップミュージックのラップ（rap）？

先生：サランラップとか、ラッピング用紙のラッパーよ。包み込む（wrap）、包み込む物（wrapper）。プリミティブ型のデータを内蔵させた、オブジェクトのことよ。

```
"abc"                 // String string "abc"
String("abc")         // String string "abc"
new String("abc")     // String object "abc"
Object("abc")         // String object "abc"
new Object("abc")     // String object "abc"

1                     // Number number 1
Number(1)             // Number number 1
new Number(1)         // Number object 1
Object(1)             // Number object 1
new Object(1)         // Number object 1

true                  // Boolean boolean true
Boolean(true)         // Boolean boolean true
new Boolean(true)     // Boolean object true
Object(true)          // Boolean object true
new Object(true)      // Boolean object true

var s; s = Symbol()   // Symbol symbol undefined
Object(s)             // Symbol object {}
new Object(s)         // Symbol object {}
```

object と表示されているものはラッパーオブジェクト

```
1.p;              // ! SyntaxError: Expected ';'
1.p = "good";     // ! SyntaxError: Expected ';'
```

「1」はプリミティブなので、プロパティを使おうとするとエラーになります。↑

```
var n; n = 1      // Number number 1
n.p = "good"      // String string "good"
n.p               // Undefined undefined undefined
```

「1」を変数に記憶すると、プロパティを付加しようとする時に、一時的にラッパーオブジェクトが作られるので、エラーになりませんが、その後、プリミティブに戻るので、プロパティの値は維持されません。↑

```
var n; n = new Number(1) // Number object 1
n.p = "good"             // String string "good"
n.p                      // String string "good"
```

ラッパーオブジェクトを陽に作れば、プロパティの書き込みや読み出しが正しく行われます。↑

```
"abc".p = "good"         // String string "good"
"abc".p                  // Undefined undefined undefined
```

数字とは異なり、文字列へのプロパティの書き込みはエラーになりませんが、値は保持されません。↑

```
var s; s = "abc"         / String string "abc"
s.p = "good"             // String string "good"
s.p                      // Undefined undefined undefined
```

文字列を変数に記憶した場合は、数字を変数に記憶した場合と同じ動作（エラーでないが保持しない）。↑

```
var s; s = new String("abc")      // String object "abc"
s.p = "good"                      // String string "good"
s.p                               // String string "good"
```

ラッパーオブジェクトを陽に作れば、プロパティの書き込みや読み出しが正しく行われます。これは、数字の場合と同じです。

なな：ふ～～ん。何に使うの？

先生：正直、あまり使いません。でも、データに注釈をぶらさげることができます。
うまく使うと、プログラムがコンパクトになったりします。

第8回　お遊び（実用性まったく無し！）（知的好奇心用）

なな：今日はお遊び？

先生：知的なお遊びを通じて、「値」の仕組みを学ぼうということです。

「じっけん」じゃあ！

```
var o; o = new Number(1)      // Number object 1
o+""                          // String string "1"
o.valueOf()                   // Number number 1
o.value = 2                   // Number number 2
o.valueOf()                   // Number number 1
o.value                       // Number number 2
```

先生：Numberオブジェクトに valueプロパティを付与しても、valueOf() に影響はありません。↑

```
var o1; o1 = { value:1, valueOf:function() { return this.value; } }
                              // Object object {"value":1}
o1+0                          // Number number 1
var o2; o2 = o1               // Object object {"value":1}
o2+0                          // Number number 1
o2.value = 2                  // Number number 2
o2+0                          // Number number 2
o1+0                          // Number number 2
```

0 を足すと 1

コピー

片方を 2 に

相手まで 2 になった！

Numberオブジェクトではない、一般的なオブジェクトに、valueプロパティを付与し、valueOf() メソッドで返すようにしてみました。でも、コピーはアドレス渡しです。↑

```
var o3; o3 = { toString: function() { return "2"; },
               valueOf: function() { return 1 } }
                              // Object object {}
o3+0                          // Number number 1
o3+""                         // String string "1"
o3.toString()                 // String string "2"
```

valueOf が使われる

toString を強制

一般的なオブジェクトに、2 を返す toString() メソッドと、1 を返す valueOf() メソッドを付与してみました。valueOf() メソッドが優先的に使われますが、toString() を明示的に使うことも可能です。↑

```
var o4; o4 = { toString:function() { return "2"; } }          // Object object {}
o4+""                         // String string "2"
```

一般的なオブジェクトに、toString() メソッドだけ付与すると、文字列のようになります。↑

```
var o5; o5 = new Number(1)    // Number object 1
o5.valueOf = function(){return 2;};       // Function function undefined
o5+0                          // Number number 2
o5                            // Number object 2
```

Numberオブジェクトに valueOf() メソッドを付与すると、元の数は完全に見えなくなります。↑

```
var o6; o6 = new Number(1)    // Number object 1
o6.valueOf = function(){return "2";} // Function function undefined
o6                            // Number object 2
o6+0                          // String string "20"
o6-0                          // Number number 2
```

Numberオブジェクト「1」に、文字列「"2"」を返す valueOf() メソッドを付与すると、String「"2"」オブジェクトのようになってしまいます。↑

```
var s5; s5 = new String("abc");          // String object "abc"
s5.valueOf=function(){return"def";} //    Function function undefined
s5                                       // String object "abc"
s5+""                                    // String string "def"
s5 == "abc"                              // Boolean boolean false
s5 == "def"                              // Boolean boolean true
[...s5]                                  // Array object ["a","b","c"]
[...(s5+"")]                             // Array object ["d","e","f"]
```

さいごの極めつけ。Stringブジェクト「"abc"」に、文字列「"def"」を返す valueOf() メソッドを付与すると、中途半端な状態になります。使い方によって、"abc" のほうが出てきたり、"def" のほうが出てきたりします。

なな：ロバとウマを掛け合わせたらラバになったみたい。

先生：先生が言いたいのは、オブジェクトを、数値や文字列のプリミティブ値に変換する方法は、valueOf、toString メソッドで決まるということです！