Default Object Methods

- contructor
  - refers to the constructor function used to create an object, returns no value
  - e.g. `function Circle(x,y,r) {`
    `this.x = x; this.y = y; this.r = r;}`
- toString()
  - automatically called for conversions to string … but may not return a useful format - JSON.stringify(obj) is often better - see example object_string.html
- toValue()
  - automatically called for conversions to numbers

Inheritance Hack: prototype

Example: prototype.html

```
function Circle(x,y,r) {
    this.x = x; this.y = y; this.r = r;
}

Circle.prototype.pi = 3.1415926534;
Circle.prototype.area = function() {
    return this.pi * this.r * this.r;
}
```

- A property not defined in the object itself will be looked up in the prototype object
- Changing a property of an object will create a local copy if the property is defined in the prototype object; useful for creating instances that differ from the standard (similar to subclassing in an OO-language like Java)

Array properties and methods

- length( )
- join(sep) // returns string of elements sep'd by sep
- reverse( ), concat(a1, a2, … aN)
- sort(func) // uses user-defined func to sort
- slice(start, [length]) // return (subarray) starting at start, if optional length is specified, treats as length of source array
- splice(index, N, elt1, … eltN)
- push(e1,…,eN) ; pop() // adds N; removes 1 element(s) from end of array
- shift() ; unshift(e1,…, eN) // removes 1; adds N elements to the front of array
- toString( )

Regular Expressions

- Example re.html

```
var p1 = new RegExp("s$");
var p2 = /s$/;
var myStr = "patterns";
if (p2.test(myStr)) { alert("plural"); }
var myStr = "pattern";
if (!p2.test(myStr)) { alert("singular"); }
```

outputs what?

- Compatible with Perl regular expression syntax
- Used in certain basic String methods
  - **search(), replace(), math(), split()**

Control Flow, Comments

- if statement:

```
if (test) {
    statements;
} else if (test) {
    statements;
} else {
    statements;
}
```

- identical structure to Java's if statement ... but JavaScript allows almost any value as a test!

- Pythoners beware:

```
if (e)
  d = c.length;
  s += c.length+2;
```

- Comments:

```
// single-line comment
/*
 * multi-line comment
 * multi-line comment
 */
```

(same as Java comment syntax)

Control Flow

- for loop (same as Java):

```
for (init; test; update) {
    statements;
}

for (var i=0; i<10; i++) {
    print(i + "\n");
}
```

- while loop:

```
while (test) {
    statements;
}

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

- break and continue keywords same as Java

JavaScript the Language – Cautions

- Implied global variables: declaring a variable without the keyword "var" won't result in an error message, but it will silently put that variable in the global scope, even if the declaration is inside a function.
  - We all know about the dangers of global variables and try not to rely on them, but JS makes it easy to create them by accident.
  - Leads to subtle, hard to track down bugs

JavaScript the Language – Cautions

- Unusual treatment of Boolean values can lead to hard-to-find bugs.
- testing for wrong falsish value can have bad results:

```
function login(user) {
    // passes with 0, "", undefined, false, ...
    if (user.name == null) { ... }
```

== produces odd results for falsish values:

```
""   == false          // true
 0   == false          // true
"0"  == false          // true
""   == '0'            // false
""   == 0              // true
null == undefined      // true
"  \t  \n " == 0       // true
```

JavaScript the Language – Cautions

- JS has a complex algorithm that allows you to omit semicolons and it will automatically insert them
  - maybe helpful (?) for bad programmers who forget to use them, but often has weird and confusing results:

```
// return an object
return
{
    name: "Joe",
    age: 15
}
```

```
// code turns into...
return;
{
    name: "Joe",
    age: 15
}
```

JavaScript the Language – Cautions

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

- "for-each" loops over each property's name in the object
  it also loops over the object's methods!
- If the object is an array, the name will be the index of the array element, not its value

```
> var ducks = ["Huey", "Dewey", "Louie"];
> for (x in ducks) { print(x); }
0
1
2
```

- Output order not predictable
- Generally considered broken; discouraged from use in most cases (in jQuery we have $.each() instead