<table>
<thead>
<tr>
<th>category</th>
<th>1960s</th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
</tr>
</thead>
<tbody>
<tr>
<td>scientific</td>
<td>Fortran</td>
<td></td>
<td></td>
<td>Matlab</td>
<td></td>
</tr>
<tr>
<td>business</td>
<td>Cobol</td>
<td>DBMSes</td>
<td>SQL</td>
<td>VB</td>
<td></td>
</tr>
<tr>
<td>functional</td>
<td>Lisp</td>
<td>ML, Scheme</td>
<td>Erlang</td>
<td>Haskell</td>
<td>F#</td>
</tr>
<tr>
<td>imperative/procedural</td>
<td>Algol</td>
<td>Pascal, C,</td>
<td>Ada, C++</td>
<td>Java</td>
<td>C#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scripting</td>
<td>BASIC</td>
<td></td>
<td>Perl</td>
<td>Python,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ruby, PHP,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JavaScript</td>
<td></td>
</tr>
<tr>
<td>logical</td>
<td></td>
<td>Prolog</td>
<td>CLP(R)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is JavaScript?

• created in 1995 by Brandon Eich of Netscape/Mozilla
  ▪ "JS had to "look like Java" only less so, be Java’s dumb kid brother or boy-hostage sidekick. Plus, I had to be done in ten days or something worse than JS would have happened." - Brandon Eich
  ▪ originally called "LiveScript" to match Netscape branding
  ▪ renamed to JavaScript to capitalize on popularity of Java
  ▪ submitted as a standard to ECMA in 1997 as "ECMAScript"

• not directly related to Java
  ▪ Eich claims he was most influenced by Self and Scheme
  ▪ some JS syntax, libraries, etc. are ripped off by Java, C
  ▪ D. Crockford: "JavaScript is Lisp in C's clothing."
JavaScript today

- possibly the most used programming language today (!!)
  - mostly used for client-side web page scripting, but increasingly used to build server apps, other programs
  - current standardized version: ECMAScript 5 (2009)

- Is JavaScript a bad programming language??
  - had bad browser behavior, slow, poor web coders, etc.
  - recent implementations are faster, better, more stable
  - JS in browser works with "DOM" (Document Object Model)
  - related JS+web technologies: Ajax, JSON, jQuery, etc.
  - spin-off languages: JScript (MS), ActionScript (Adobe), etc.
JavaScript vs. Java

- interpreted, not compiled
  - dynamic typing
  - first-class functions; nested functions; closures
  - a structured, imperative object-oriented, scripting language
  - prototype-based object and inheritance system
  - sophisticated first-class resizable array type
  - first-class regular expression support

- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't always need to be declared
  - key construct is first-class function rather than the class
Running JS code in a browser

```
<html>
  <head>
    <script src="myfile.js" type="text/javascript"></script>
  </head>
  <body>
    <p>My web page</p> ...
  </body>
</html>
```

- We won't be doing this!
  - aside: Firebug extension
Running JS without a browser

- **CommonJS**: project started in 2009 to create a standard library of JS types and functions for all non-web apps
  - Rhino (Mozilla)
  - V8 (Google / Chrome)
  - Narwhal
  - others: Ringo, Joyent, Sprout, Persevere

- We support the **Rhino** runtime for this course.
  - `java -jar rhino.jar JSFileName`
The Rhino debugger

```java
java -classpath rhino.jar
   org.mozilla.javascript.tools.debugger.Main filename.js
```

- [http://www.mozilla.org/rhino/debugger.html](http://www.mozilla.org/rhino/debugger.html)
JavaScript syntax
print (CommonJS)

print(expr, expr, ..., expr);

- provided by Rhino as part of CommonJS
  - print("Hello, world!\n");
  - print(1+1, 4, 3*2); // 2 4 6

- other shell variables/functions:
  - arguments, environment, help, defineClass, deserialize, load(filename), loadClass, readFile(name), readURL, runCommand, seal, serialize, spawn, sync, quit, version
- doesn't work in web browsers (use alert instead)
**Variables**

```javascript
var name = expression;
```

- Examples:
  - ```javascript
    var age = 32;
    var weight = 127.4;
    var clientName = "Connie Client";
  ```

- variables are declared with `var` keyword (case sensitive)
- types not specified, but JS does have types
  - `Number`, `Boolean`, `String`, `Array`, `Object`, `Function`, `Null`, `Undefined`
  - can find out a variable's type by calling `typeof`
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);

• integers and real numbers are the same type
  ▪ (no int vs. double)

• same operators: + - * / % ++ -- = += -= *= /= %=
  ▪ similar precedence to Java
  ▪ many operators auto-convert types: "2" * 3 is 6
## Number properties/methods

### Number object "static" properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number.MAX_VALUE</td>
<td>largest possible number, roughly $10^{308}$</td>
</tr>
<tr>
<td>Number.MIN_VALUE</td>
<td>smallest <em>positive</em> number, roughly $10^{-324}$</td>
</tr>
<tr>
<td>Number.NaN</td>
<td>Not-a-Number; result of invalid computations</td>
</tr>
<tr>
<td>Number.POSITIVE_INFINITY</td>
<td>infinity; result of $1/0$</td>
</tr>
<tr>
<td>Number.NEGATIVE_INFINITY</td>
<td>negative infinity; result of $-1/0$</td>
</tr>
</tbody>
</table>

### Number instance methods

- `.toString([base])`: convert a number to a string with optional base
- `.toFixed(digits)`: fixed-point real with given # digits past decimal
- `.toExponential(digits)`: convert a number to scientific notation
- `.toPrecision(digits)`: floating-point real, given # digits past decimal

### Global methods related to numbers

- `isNaN(expr)`: true if the expression evaluates to NaN
- `isFinite(expr)`: true if `expr` is neither NaN nor an infinity
The Math object

```javascript
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

- Math methods: abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan
- properties: E, PI
# Math properties/methods

<table>
<thead>
<tr>
<th>Method/Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.E</td>
<td>( e ), base of natural logarithms: 2.718...</td>
</tr>
<tr>
<td>Math.LN10, Math.LN2, Math.LOG2E, Math.LOG10E</td>
<td>natural logarithm of 10 and 2; logarithm of ( e ) in base 2 and base 10</td>
</tr>
<tr>
<td>Math.PI</td>
<td>( \pi ), circle's circumference/diameter: 3.14159...</td>
</tr>
<tr>
<td>Math.SQRT1_2, Math.SQRT2</td>
<td>square roots of ( \frac{1}{2} ) and 2</td>
</tr>
<tr>
<td>Math.abs(n)</td>
<td>absolute value</td>
</tr>
<tr>
<td>Math.acos/asin/atan(n)</td>
<td>arc-sin/cosine/tangent of angle in radians</td>
</tr>
<tr>
<td>Math.ceil(n)</td>
<td>ceiling (rounds a real number up)</td>
</tr>
<tr>
<td>Math.cos/sin/tan(n)</td>
<td>sin/cosine/tangent of angle in radians</td>
</tr>
<tr>
<td>Math.exp(n)</td>
<td>( e^n ), ( e ) raised to the ( n )th power</td>
</tr>
<tr>
<td>Math.floor(n)</td>
<td>floor (rounds a real number down)</td>
</tr>
<tr>
<td>Math.log(n)</td>
<td>natural logarithm (base ( e ))</td>
</tr>
<tr>
<td>Math.max/min(a, b...)</td>
<td>largest/smallest of 2 or more numbers</td>
</tr>
<tr>
<td>Math.pow(x, y)</td>
<td>( x^y ), ( x ) raised to the ( y )th power</td>
</tr>
<tr>
<td>Math.random()</td>
<td>random real number ( k ) in range ( 0 \leq k &lt; 1 )</td>
</tr>
<tr>
<td>Math.round(n)</td>
<td>round number to nearest whole number</td>
</tr>
<tr>
<td>Math.sqrt(n)</td>
<td>square root</td>
</tr>
</tbody>
</table>
Comments (same as Java)

// single-line comment

/*
multi-line comment
multi-line comment
*/

• (identical to Java's comment syntax)
var s = "Connie Client";
var firstName = s.substring(0, s.indexOf(" "));
var len = s.length;       // 13
var s2 = 'Melvin Merchant'; // can use "" or '

• String methods: `charAt`, `charCodeAt`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`
  ▪ `charAt` returns a one-letter string (there is no `char` type)
  ▪ `length` is a property (not a method as in Java)

• concatenation with `+`: `1 + 1` is 2, but "1" + 1 is "11"
• strings can be `compared` with `<`, `<=`, `==`, `!=`, `>`, `>=`
# String methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String.fromCharCode(expr)</code></td>
<td>Converts ASCII integer → String</td>
</tr>
<tr>
<td><code>.charAt(index)</code></td>
<td>Returns character at index, as a String</td>
</tr>
<tr>
<td><code>.charCodeAt(index)</code></td>
<td>Returns ASCII value at a given index</td>
</tr>
<tr>
<td><code>.concat(str...)</code></td>
<td>Returns concatenation of string(s) to this one</td>
</tr>
<tr>
<td><code>.indexOf(str[, start])</code></td>
<td>First/last index at which given string begins in this string, optionally starting from given index</td>
</tr>
<tr>
<td><code>.lastIndexOf(str[, start])</code></td>
<td>First/last index at which given string begins in this string, optionally starting from given index</td>
</tr>
<tr>
<td><code>.match(regexp)</code></td>
<td>Returns any matches for this string against the given string or regular expression (&quot;regex&quot;)</td>
</tr>
<tr>
<td><code>.replace(old, new)</code></td>
<td>Replaces first occurrence of old string or regular expr. with new string (use regex to replace all)</td>
</tr>
<tr>
<td><code>.search(regexp)</code></td>
<td>First index where given regex occurs</td>
</tr>
<tr>
<td><code>.slice(start, end)</code></td>
<td>Substr. from start (inclusive) to end (exclusive)</td>
</tr>
<tr>
<td><code>.substring(start, end)</code></td>
<td>Substr. from start (inclusive) to end (exclusive)</td>
</tr>
<tr>
<td><code>.split(delimiter[, limit])</code></td>
<td>Break apart a string into an array of strings</td>
</tr>
<tr>
<td><code>.toLowerCase()</code></td>
<td>Return new string in all upper/lowercase</td>
</tr>
<tr>
<td><code>.toUpperCase()</code></td>
<td>Return new string in all upper/lowercase</td>
</tr>
</tbody>
</table>
More about Strings and numbers

- escape sequences behave as in Java: `\' \" \& \n \t \n` \n
- convert string to number with `parseInt`, `parseFloat`:
  ```javascript
  var count = 10;
  var s1 = "" + count; // "10"
  var s2 = count + " bananas, ah ah ah!";
  var n1 = parseInt("42 is the answer"); // 42
  var n2 = parseInt("0x2A", 16); // 42
  var n3 = parseFloat("3.1415"); // 3.1415
  var bad = parseInt("booyah"); // NaN
  ```

- access the letters of a String with `[ ]` or `charAt`:
  ```javascript
  var firstLetter = s[0];
  var firstLetter = s.charAt(0);
  var lastLetter = s.charAt(s.length - 1);
  ```
for (initialization; test; update) {
  statements;
}

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

var s1 = "hi, there!!!", s2 = "";
for (var i = 0; i < s1.length; i++) {
  var c = s1.charAt(i);
  if (c >= "a" && c <= "z") {
    s2 += c + c;
  }
}
Logical operators

>  <  >=  <=  &&  ||  !  ==  !=  ===  !==

- most logical operators automatically convert types:
  - `5 < "7"` is true
  - `42 == 42.0` is true
  - "5.0" == 5 is true

- `===`, `!==` are strict equality tests; checks type and value
  - "5.0" === 5 is false
The if/else statement

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

• identical structure to Java's if/else statement...
  ▪ but JavaScript allows almost any value as a test!
Boolean type

var iLike341 = true;
var ieIsGood = "IE6" > 0; // false
if ("JS is great") { ... } // true
if (0 || "") { ... } // false

• any value can be used as a test
  ▪ "falsey" values: 0, 0.0, NaN, "", null, and undefined
  ▪ "truthy" values: anything else

• converting a value into a boolean explicitly:
  var boolValue = Boolean(otherValue);
  var boolValue = !!(otherValue);
&& and || in depth

• a && b is a binary operator that returns:
  ▪ if a is truthy, then b, else a
  ▪ *(this turns out to be a truthy/falsey value in the right cases)*

• a || b is a binary operator that returns:
  ▪ if a is truthy, then a, else b
  ▪ *(this turns out to be a truthy/falsey value in the right cases)*

• Examples:
  ▪ 0 || 42 || 12 || -1 returns 42 (truthy)
  ▪ NaN || null || "" returns "" (falsey)
  ▪ 1 + 1 && 6 && 9 returns 9 (truthy)
  ▪ 3 && 4 && null && 5 && 6 returns null (falsey)
null vs. undefined

```javascript
var ned = null;
var benson = 9;
var caroline;
```

- at this point in the code:
  - ned is null
  - benson is 9
  - caroline is undefined

- undefined: has not been declared, does not exist
- null: exists, but specifically assigned an empty value
  - Why does JavaScript have both of these?
while (test) {
    statements;
}

do {
    statements;
} while (test);

• break and continue keywords also behave as in Java
function name(paramName, ..., paramName) {
    statements;
}

function myFunction(name) {
    print("Hello, " + name + "!
");
    print("How are you\n");
}

- unlike in Java, functions are first-class (can be stored as variables, passed as parameters, returned, ...)

JavaScript keywords

- break  case  catch  continue  debugger
- default  delete  do  else  finally
- for  function  if  in  instanceof
- new  return  switch  this  throw
- try  typeof  var  void  while
- with

- Reserved words (these don't do anything yet):
  - class  const  enum  export  extends
  - import  implements  interface  let  package
  - private  protected  public  static  super  yield