CSC 470 – Section 3

Topics in Computer Science: Advanced Browser Technologies

Mark F. Russo, Ph.D.

Spring 2016

Lecture 2

Eloquent JavaScript: Chapters 1 & 4

JavaScript

- Dynamic programming language for general-purpose application
- Supports a type of object-oriented, imperative, and functional programming styles
- Embedded in nearly all modern web browsers
- Means by which all web browser technologies may be accessed

- A "brackets" language, like C++ and Java
- Curley brackets used to delineate code blocks "{ … }"
- Statements end with a semicolon ";"

Values, Types, and Operators

Six basic types of values in JavaScript

1. Numbers
2. Strings
3. Booleans
4. Objects
5. Functions
6. *undefined values*

- JavaScript numbers are always 64-bit floating point values

Data Type Literals

`Boolean` - one of two keywords { true, false }

- Example values: `true, false`

`Number` - number with or without decimal places

- Example values: `0.0, 3.14159, -2, 255, 31039`
- Range: `~ -1.798E+308 to ~ 1.798E+308`
  `~ -5E-324 to ~ 5E-324`
- Special Numbers: `Infinity, -Infinity, NaN`

`String` - sequence of char enclosed by double or single quotes

- Example values:

```
"Fred", '123' , ""
"This String is defined over \
 multiple lines"
```

Data Type Literals

C:\TCNJ\2016Spring\CSC470\Lectures\02\literals.js

File Edit View Execute Environment Help

Open File... Save Save As... Run Inspect Display

```
1  // Number literals
2  console.log( 3145926 );
3  console.log( 3.145926 );
4  console.log( 3145926e-6 );
5  console.log( 0xFF );      // Hexadecimal
6
7  // Boolean literals
8  console.log( true );
9  console.log( false );
10
11 // String literals
12 console.log( "Hello World" );
13 console.log( 'TCNJ' );
14
15 // Infinity
16 console.log( 100 / 0 );
17
18 // NaN
19 console.log( 100/"A" );
20
```

Line 20, Col 1

Console - http://www.w3schools.com/js/js_numbers.asp

In... C... De... Styl... Perfor... N...

Net CSS JS Security Logging Server Clear

3145926	Scratchpad/6:2:1
3.145926	Scratchpad/6:3:1
3.145926	Scratchpad/6:4:1
255	Scratchpad/6:5:1
true	Scratchpad/6:8:1
false	Scratchpad/6:9:1
Hello World	Scratchpad/6:12:1
TCNJ	Scratchpad/6:13:1
Infinity	Scratchpad/6:16:1
NaN	Scratchpad/6:19:1

Number Literals

Decimal Notation - With decimal places

- `1.23, 3.1415926`

Decimal notation - Without decimal places

- `123, 255`

Decimal notation - Exponential notation

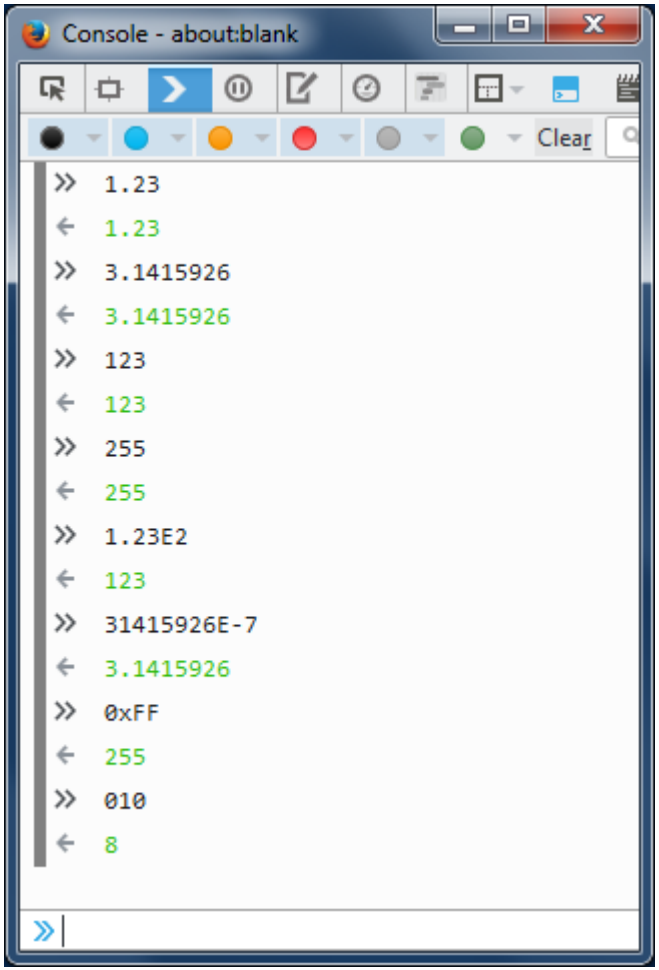
- `1.23E2, 31415926E-7`

Hexadecimal notation - Begins with `0x`

- `0xFF // 255`

Octal notation - Begins with zero

- `010 // 8`

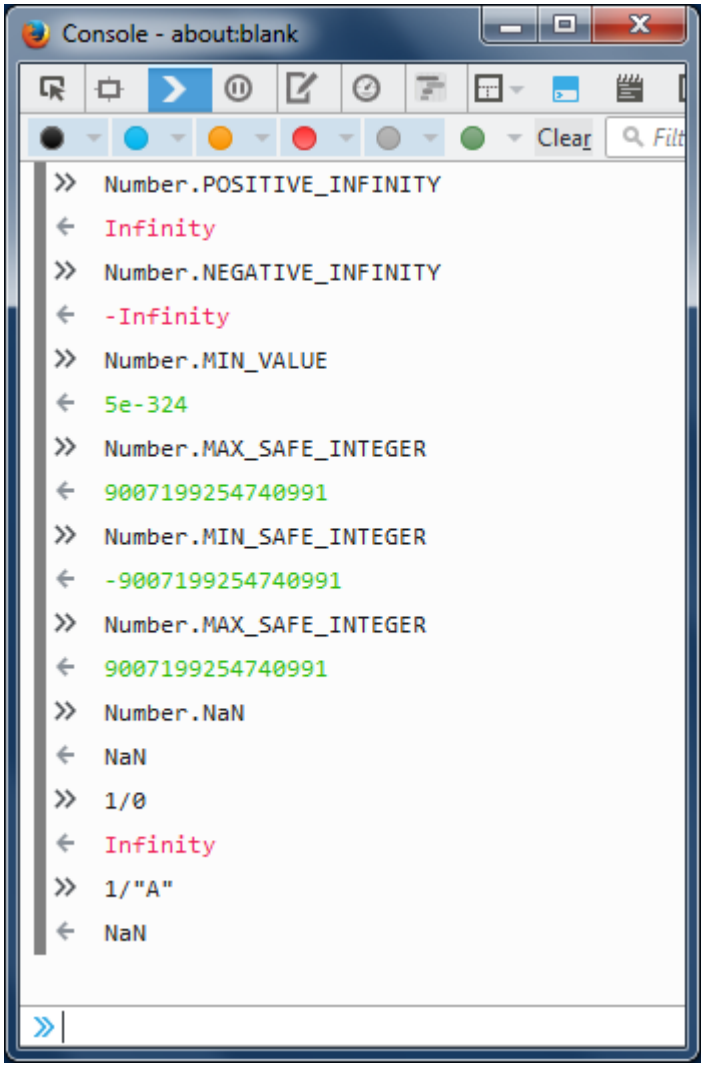


Regardless of the number format, internally JavaScript stores all numbers as double-precision floating-point values, according to the IEEE 754 Standard

Number Constants

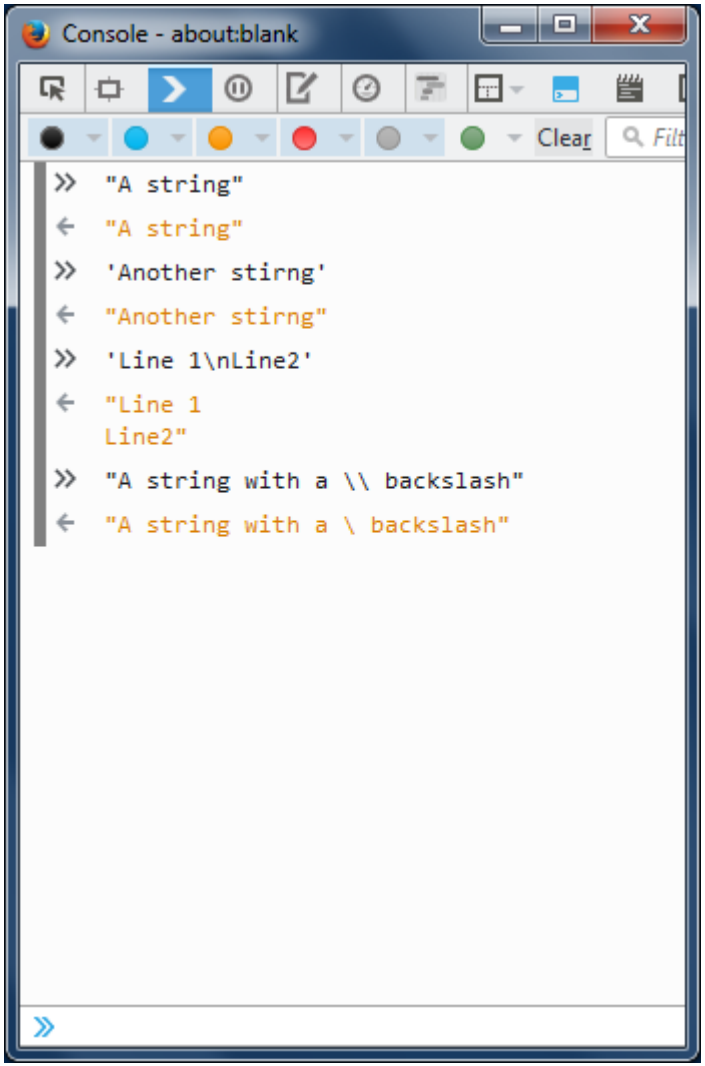
A built-in Number object provides several useful "static" constants

- `POSITIVE_INFINITY`
- `NEGATIVE_INFINITY`
- `MIN_VALUE`
- `MAX_VALUE`
- `MIN_SAFE_INTEGER`
- `MAX_SAFE_INTEGER`
- `NaN`



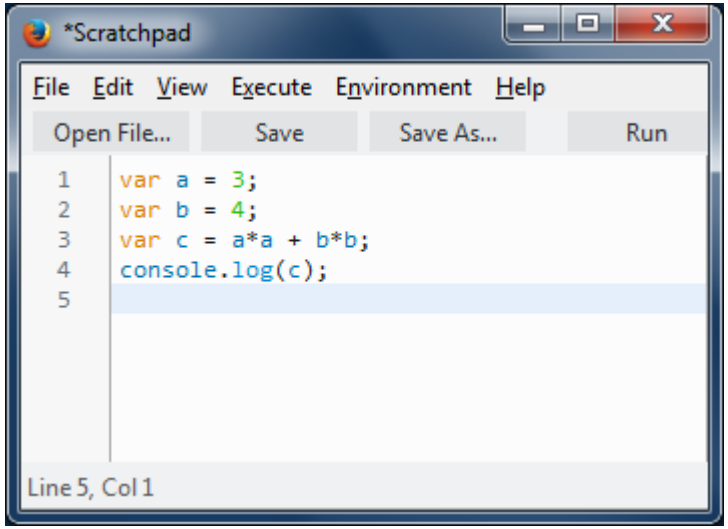
String Literals

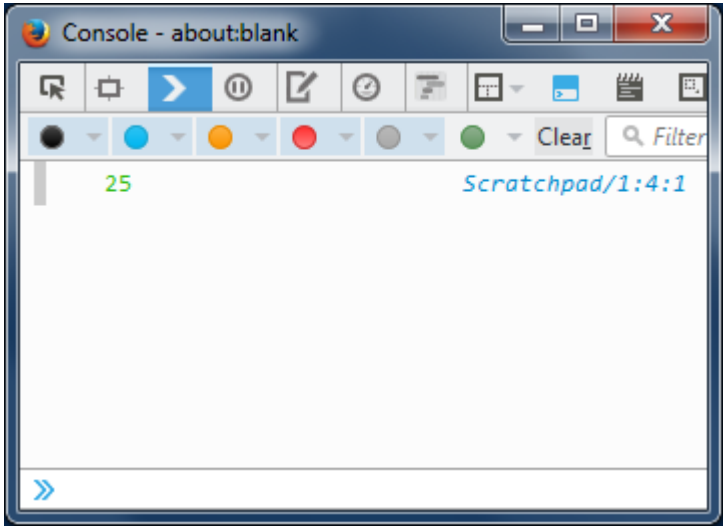
- May be created with pairs of double or single quotes
- May escape (\) to insert quotes or other special symbols
- May use "\" at the end of a line to continue defining over multiple lines
    - Note that a \ does not insert a newline, use `\n` to do that.



Variables

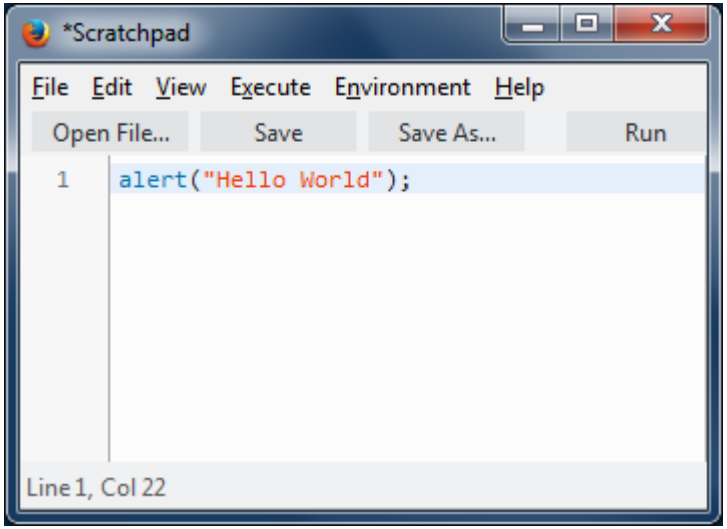
- New variables are declared using the `var` keyword
  - Variables are not typed, data values have type.
  - All variables are declared with the `var` keyword.
- Standard variable naming rules apply
  - Variable names must begin with a letter, `$` or _
  - Variable names are limited to numbers, digits `$` and _

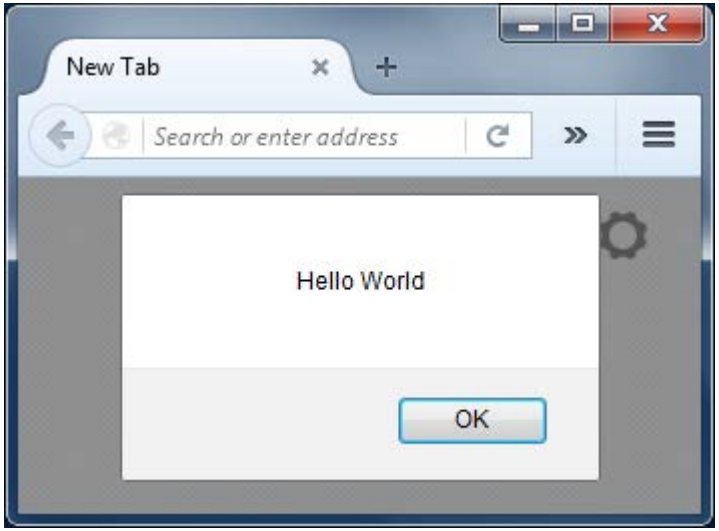




`alert()` Browser Method

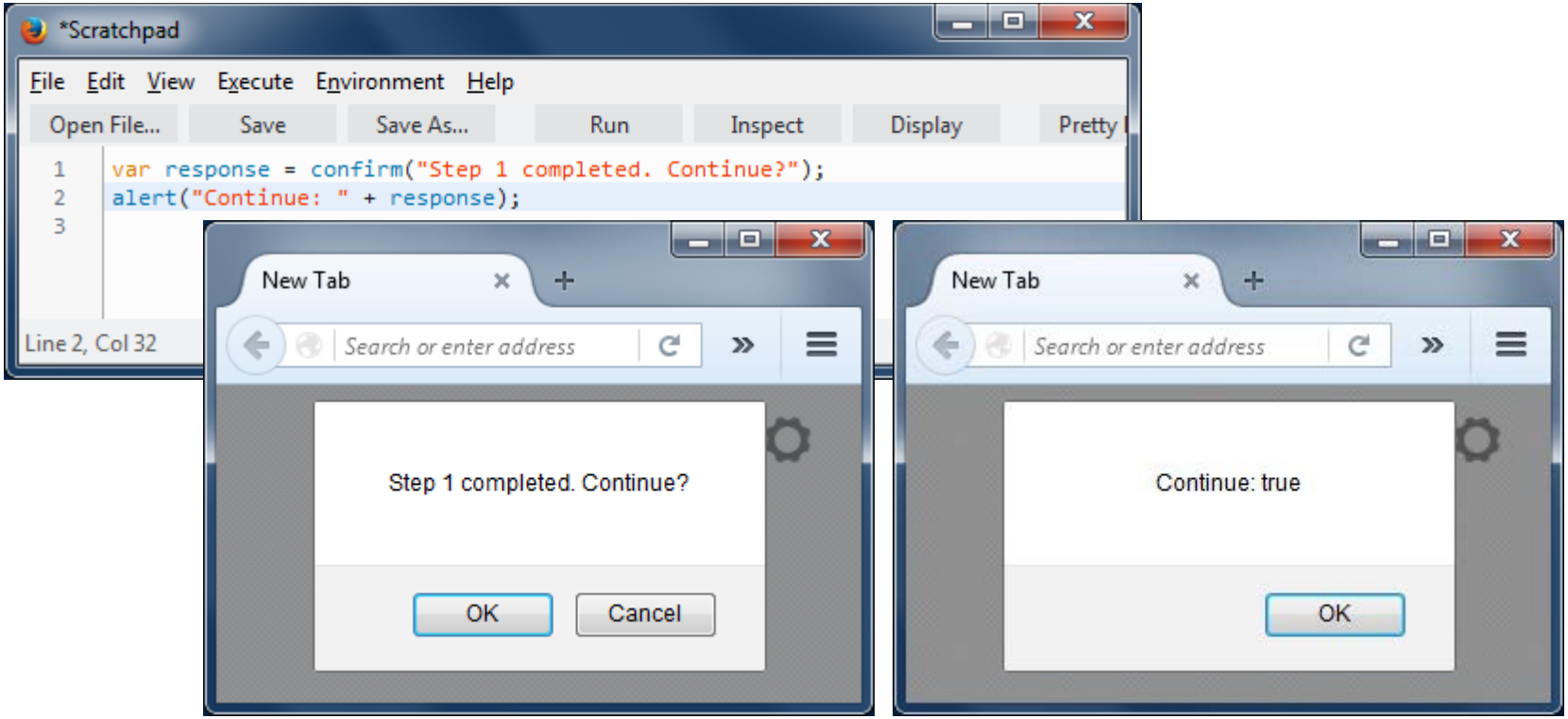
- Built-in browser window object function that displays a message in a modal window.
- An [OK] button is used to dismiss the window.
- Useful for communicating a message to the user of a browser.





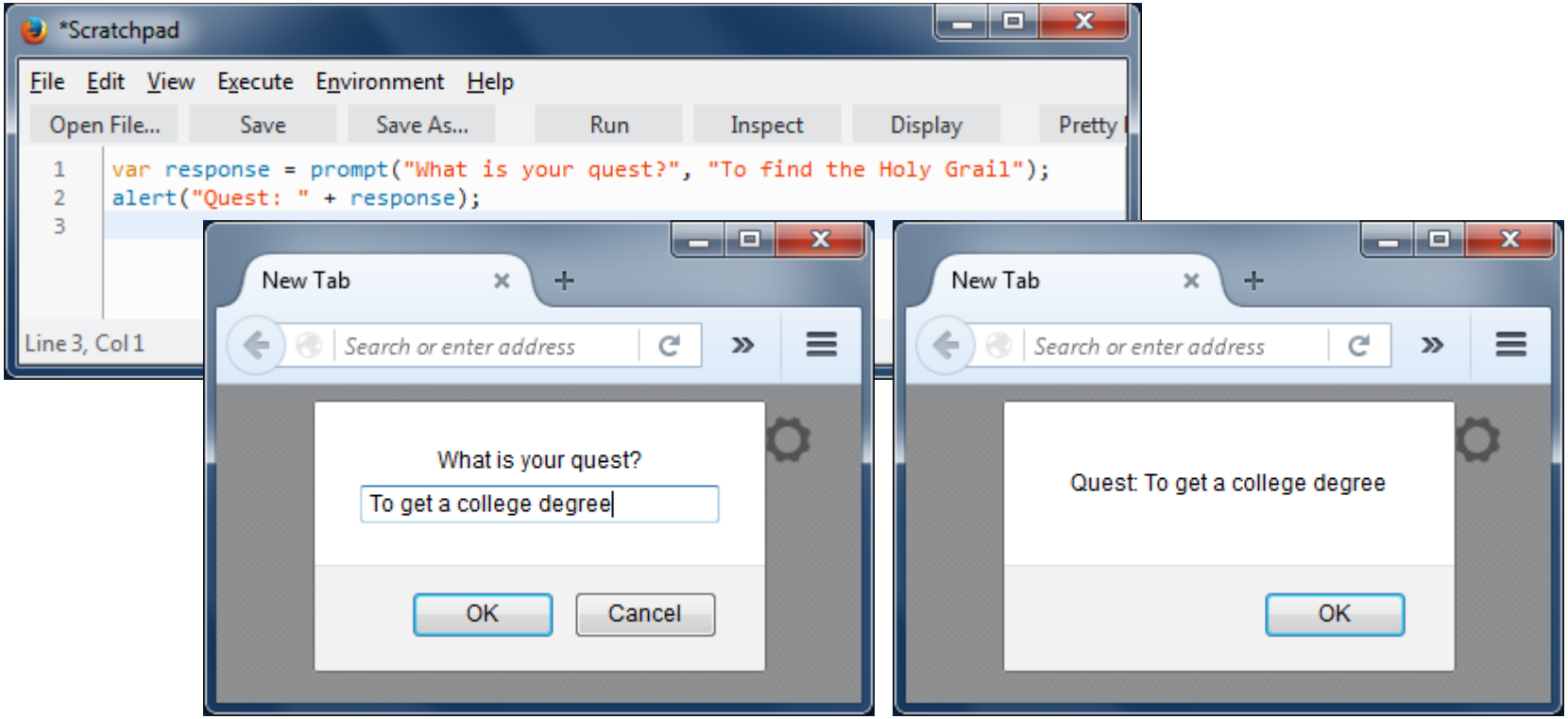
`confirm()` Browser Method

- Built-in browser window object function that displays a message in a modal window and accepts a response from the user
- If the [OK] button is clicked `true` is returned.
- If the [Cancel] button is clicked `true` is returned.
- Useful for confirming an action with the browser user.



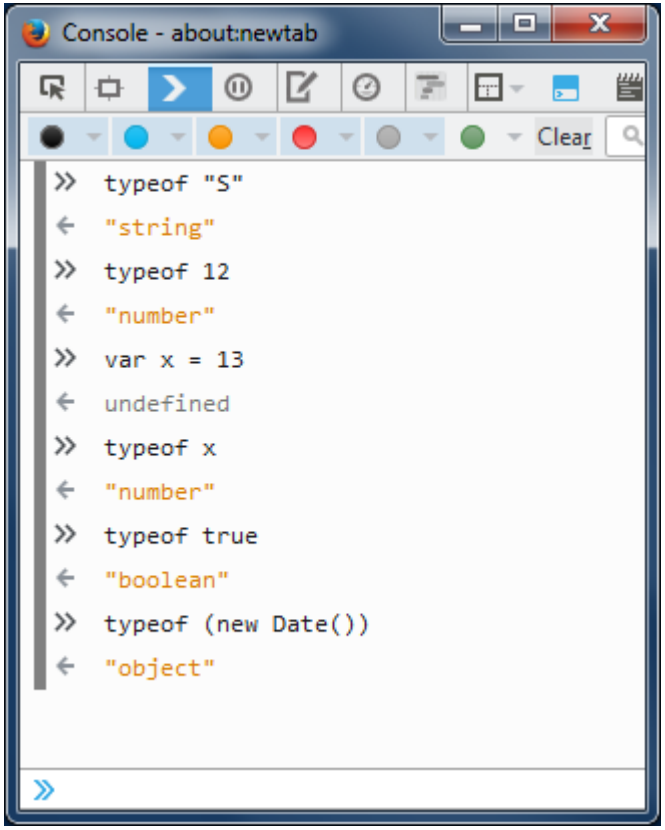
`prompt()` Browser Method

- Built-in browser window object function that displays a message in a modal window and provides a way for the user to respond.
- An optional second argument provides default text.
- Clicking [OK] returns the string in the provided text box. Clicking [Cancel] returns `null`.
- Useful for collecting feedback from the browser user.



typeof

- Prefix operator to get object type
- Returns a String with type name



"use strict";

- Add string literal `"use strict";` to top of JavaScript file
- Designed to be compatible (ignored by) older versions of JavaScript
- Introduced in ECMAScript version 5
- Requires that all variables be declared
- In non-strict mode, variables not declared become globals

```
function first() {
  x = 12;
}

function second() {
  console.log(x);
}

first();
second();

// Prints 12
```

```
function first() {
  var x = 12;
}

function second() {
  console.log(x);
}

first();
second();

// Error
```

```
"use strict";
function first() {
  var x = 12;
}

function second() {
  console.log(x);
}

first();
second();

// Error
```

Undefined Values

`undefined`

- A predefined global variable
- Indicates absence of value
- Returned when variable has not yet been defined
- `typeof undefined` → `"undefined"`

`null`

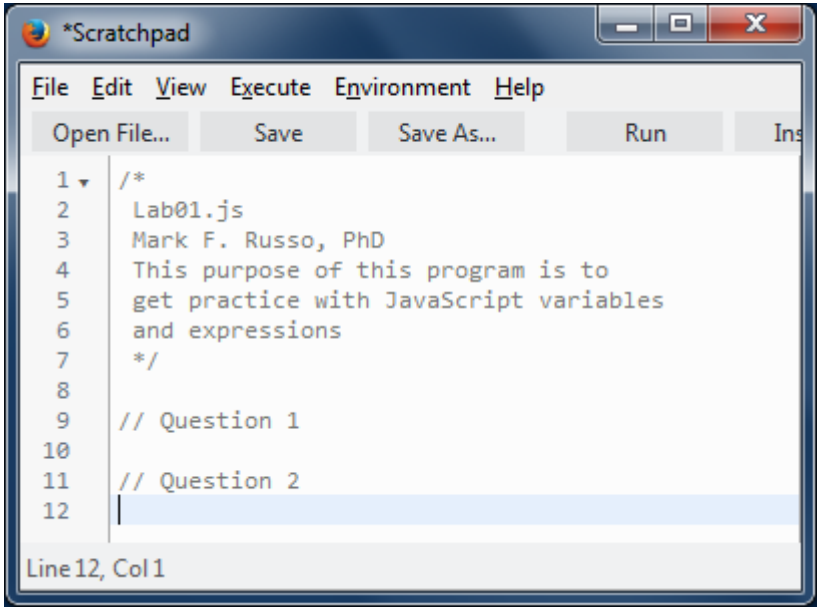
- JavaScript keyword
- Indicates absence of value
- Can be thought of as a special object value that indicates “no object”
- `typeof null` → `"object"`

Comments

- Statements used to describe a program
- Ignored by the compiler

- Two styles:

  1. Anything delimited by /* and */
  2. Anything to the right of // through the end of a line



Expressions

- An organized series of literals, variables, operators, functions and method calls
- Upon evaluation produces a single value (numeric or other)
- Follows standard mathematical expression syntax

```
-2*a*b - Math.cos(theta)
```

Operators

Symbols that transform operands (subexpressions)

- Prefix, Infix, Postfix

Mathematical `(+ - * / ...)`

- Standard mathematical operations
- PEMDAS

Relational `(< <= > >= == != ...)`

- Test relationship between related expressions.
- Always returns a boolean value (true or false).

Logical `(&& || !)`

- Logical conjunction (and), disjunction (or), negation (not).
- Always returns a boolean value (true or false).

Mathematical Operators

```
+, -, *, /
```

`i % 3;`	equivalent to	`remainder of i/3`
`i++;`	equivalent to	`i = i + 1;`
`i += 2;`	equivalent to	`i = i + 2;`
`i--;`	equivalent to	`i = i - 1;`
`i -= 3;`	equivalent to	`i = i - 3;`
`i *= 2;`	equivalent to	`i = i * 2;`
`i /= 4;`	equivalent to	`i = i / 4;`

String Concatenation

Operators may also operate on non-numeric types

'+' operates on Strings by concatenating them together

Examples:

```
"A" + "01"  -> "A01"
"one" + " " + "two" + " " + "three"  -> "one two three"
```

Modulus Operator (%)

- The "remainder operator"
- Returns the remainder after operands are divided
- Sign of result is sign of numerator

```
 4 % 2     -> ?
 5 % 2     -> ?
-5 % 2     -> ?
-6 % 3     -> ?
65 % 60    -> ?
73 % 24    -> ?
```

Relational Operators

- Compare values
- Always returns a boolean (`true` or `false`)

`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	is equivalent to
`!=`	is not equivalent to

Logical Operators

& & **logical conjunction** (and)

both arguments must evaluate to 'true' for complete expression to evaluate to 'true', otherwise 'false'

| | **logical disjunction** (or)

if either argument evaluates to 'true' the complete expression evaluates to 'true', otherwise 'false'

! **logical negation** (not)

turns 'true' to 'false' and 'false' to 'true'

Logical Operators

Conjunction (and)

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

Disjunction (or)

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

Negation (not)

A	!A
false	true
true	false

Operator Precedence

Follows well-defined rules (PEMDAS)

Precedence Level	Operator	Operation	Associates
1	+	unary plus	R to L
	-	unary minus	
2	*	multiplication	L to R
	/	division	
	%	modulus	
3	+	addition/concatenation	L to R
	-	subtraction	
4	=	assignment	R to L

Automatic Type Conversion

- JavaScript goes out of its way to accept almost any expression you give it and try to guess what you want to do.
- When an operator is applied to the “wrong” type of value, JavaScript will quietly convert that value to a type that is valid, using a set of rules
- This type coercion is often <u>not</u> what you want or expect
- This is a mistake in the design of JavaScript (in my opinion)

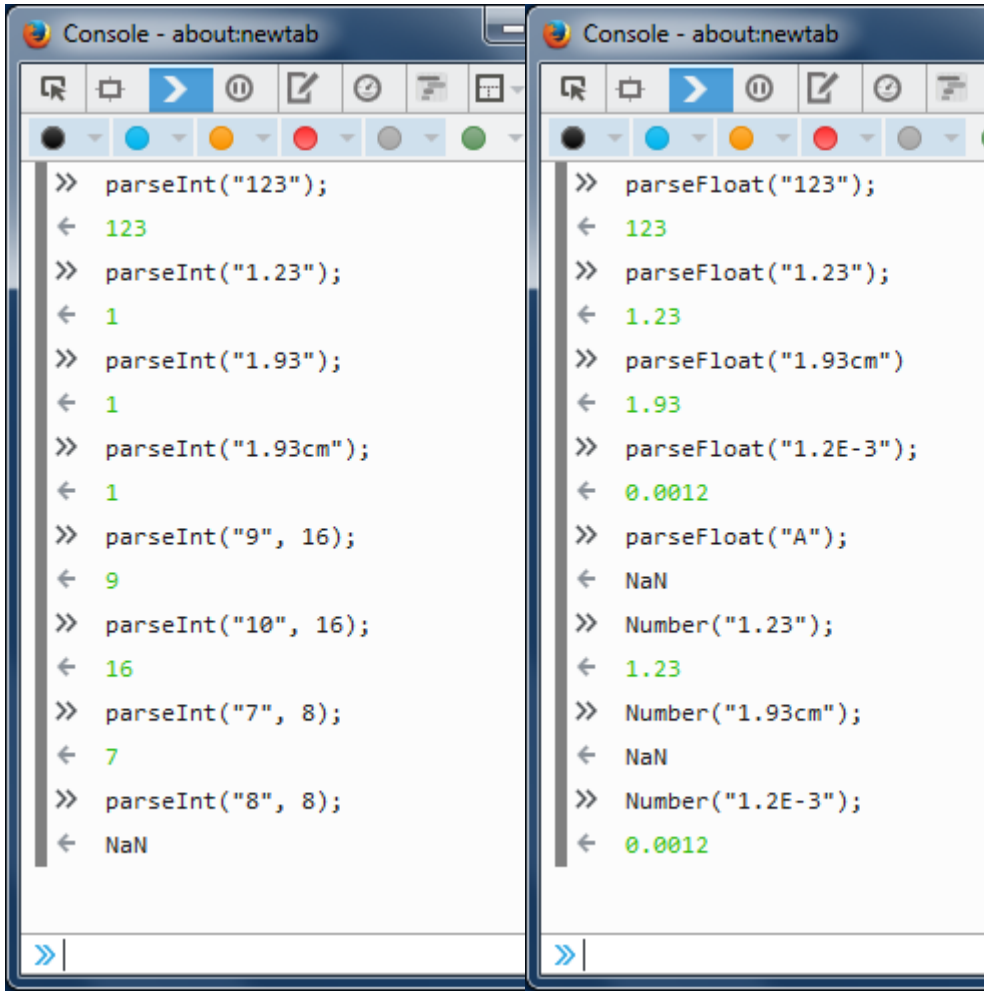
```
>> "2" + 1
"21"
>> "2" - 1
1
>> "2" * null
0
>> 2 * false
0
>> 2 / false
Infinity
```

It is easy to write a "working" program that has serious errors. Be careful.

Explicit conversion to a Number

Three global functions are available to convert values to a Number

- `parseInt(val, radix)`
  - Parse `val` and return an integer Number in base `radix`
  - Ignores non-numeric trailing characters
- `parseFloat(val)`
  - Parse `val` and return a floating point Number
  - Ignores non-numeric trailing characters
- `Number(val)`
  - Parse `val` and return a Number
  - All characters must be part of the Number



== vs. ===

- Operators === and !== are called <u>strict equality</u> operators
- Operators == and != are very loose
  - Test the "truthiness" of a value
  - Never use these operators

- JavaScript editors will often complain if you use == or !=

Equality Table (==)

	false	0	""	[[]]	[]	"0"	[0]	[1]	"1"	1	true	-1	"-1"	null	undefined	Infinity	-Infinity	"false"	"true"	{}	NaN
false	■	■	■	■	■	■	■														
0	■	■	■	■	■	■	■														
""	■	■	■	■	■																
[[]]	■	■	■																		
[]	■	■	■																		
"0"	■	■				■	■														
[0]	■	■				■															
[1]									■	■	■										
"1"								■	■	■	■										
1								■	■	■	■										
true								■	■	■	■										
-1												■	■								
"-1"												■	■								
null														■	■						
undefined														■	■						
Infinity																■					
-Infinity																	■				
"false"																		■			
"true"																			■		
{}																					
NaN																					

https://dorey.github.io/JavaScript-Equality-Table/
http://algorithmicassertions.com/visualization/2014/03/27/Better-JS-Equality-Table.html

Built-in Objects

- Arguments
- **Array**
- *Boolean*
- **Date**
- Function
- JSON
- **Math**
- *Number*
- **Object**
- RegExp
- **String**

- Error
- EvalError
- RangeError
- ReferenceError
- SyntaxError
- TypeError
- URIError

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Math Object

- Provides a large number of mathematical functions

  `sin(), cos(), abs(), pow(), ...`

  `min(), max(), random(), ...`

- All functions are "static," must be scoped with Math object name
- Also includes constants `PI`, `E` and others

```
var angle = Math.cos( Math.PI );
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

String Object

Creating…

- String literal

```
var name = "Bart";
```

- String constructor

```
var name = new String();
var name = new String("Lisa");
```

- String function

```
var name = String(123);    // "123"
```

String Object – Properties and Methods

Methods

- `charAt(…)`
- `charCodeAt(…)`
- `concat(…)`
- `contains(…)`
- `endsWith(…)`
- `includes(…)`
- `indexOf(…)`
- `replace(…)`
- `search(…)`
- `slice(…)`
- `split(…)`
- `startsWith(…)`
- `substr(…)`
- `substring(…)`
- `toLowerCase(…)`
- `toUpperCase(…)`
- `trim(…)`
- `trimRight(…)`
- `trimLeft(…)`

Properties

- `length`

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

String Object Examples

C:\TCNJ\2016Spring\CSC470\Lectures\02\string_methods.js

File Edit View Execute Environment Help

Open File... Save Save As... Run Inspect Display

```
var s = "ABCD";
console.log( s.charAt(1) );          // "B"
console.log( s.toLowerCase() );      // "abcd"
console.log( s.includes("BC") )      // true
console.log( s.substr(1, 2) );       // "BC"
console.log( s.substring(1, 3) );    // "BC"
console.log( " ABC ".trim() );       // "ABC"
console.log( "A,B,C".split(","));    // ["A", "B", "C"];
```

Line 9, Col 1

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Primitives vs. Objects: String, Number, Boolean

- Primitives are created using primitive notation or calling the Class in a non-constructor context
- Objects are created using the constructor
- `typeof` operator returns different values
- JavaScript automatically converts primitives to objects so that object methods may be used on primitives

C:\TCNJ\2016Spring\CSC470\Lectures\02\lit_vs_obj.js

File Edit View Execute Environment Help

Open File... Save Save As... Run Inspect

```
var s_lit1 = "ABC";
var s_lit2 = String("ABC");
var s_obj  = new String("ABC");

var n_lit1 = 123;
var n_lit2 = Number(123);
var n_obj  = new Number(123);

var b_lit1 = true;
var b_lit2 = Boolean(true);
var b_obj  = new Boolean(true);

console.log(typeof s_lit1);          // "string"
console.log(typeof s_lit2);          // "string"
console.log(typeof s_obj);           // "object"

console.log(typeof n_lit1);          // "number"
console.log(typeof n_lit2);          // "number"
console.log(typeof n_obj);           // "object"

console.log(typeof b_lit1);          // "boolean"
console.log(typeof b_lit2);          // "boolean"
console.log(typeof b_obj);           // "object"

console.log( s_lit1.toLowerCase() ); // "abc"
console.log( s_lit2.toLowerCase() ); // "abc"
console.log( s_obj.toLowerCase() );  // "abc"
```

Line 28, Col 1

Date Object

- Holds a date-time value
- Has no literal syntax – must be created with a constructor invoked in function or constructor contexts
- Constructors include:

```
// Returns the current date-time
new Date();

// Build date-time using number of milliseconds
// since January 1, 1970
new Date(milliseconds);

// Parses a date string in a standard syntax
new Date(dateString);

// Builds a date from date and time elements
new Date(year, month[, day[, hour[, minutes[,
         seconds[, milliseconds]]]]]);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Date Object

- `Date.now()`
- `Date.parse()`
- `get/setDate(…)`
- `get/setDay(…)`
- `get/setFullYear(…)`
- `get/setHours(…)`
- `get/setMinutes(…)`
- `get/setMilliseconds(…)`
- `get/setMonth(…)`
- `get/setSeconds(…)`
- `get/setTime(…)`
- `get/setTimezoneOffset(…)`
- `get/setYear(…)`
- `toDateString …)`
- `toGMTString(…)`
- `toISOString(…)`
- `toJSON(…)`
- `toLocaleDatestring(…)`
- `toLocaleFormat(…)`
- `toLocaleString(…)`
- `toLocaleTimeString(…)`
- `toSource(…)`
- `toTimeString(…)`
- `toUTCString(…)`

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Date Object

- `getTime(millis)` and `setTime(millis)` get and set the date-time using milliseconds since the Unix Epoch
- `Date.parse(…)` takes a date string and returns milliseconds since the Unix Epoch
- A date string representing follows RFC2822 or ISO 8601 date

C:\TCNJ\2016Spring\CSC470\Lectures\02\dates.js

File Edit View Execute Environment Help

Open File... Save Save As... Run Inspect Display

```
var s1 = "Jan 1, 2016";
var date1 = new Date(s1);
console.log(date1);

// Parse the date string and return milliseconds
var millis = Date.parse(s1);
console.log(millis);

// Get milliseconds from the data using getTime()
millis = date1.getTime();
console.log(millis);

// Add one hour of milliseconds to millis
millis += 60*60*1000;

// Set a new date using milliseconds
date1.setTime(millis);
console.log(date1);
```

Line 19, Col 1

Console - about:newtab

Clear Filter output

```
Date 2016-01-01T05:00:00.000Z        Scratchpad/5:3:1
1451624400000                        Scratchpad/5:7:1
1451624400000                        Scratchpad/5:11:1
Date 2016-01-01T06:00:00.000Z        Scratchpad/5:18:1
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

RFC 2822 | 3.3. Date and Time Specification

```
date-time   = [ day-of-week "," ] date FWS time [CFWS]
day-of-week = ([FWS] day-name) / obs-day-of-week
day-name    = "Mon" / "Tue" / "Wed" / "Thu" / "Fri" / "Sat" / "Sun"
date        = day month year
year        = 4*DIGIT / obs-year
month       = (FWS month-name FWS) / obs-month
month-name  = "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" /
              "Jul" / "Aug" / "Sep" / "Oct" / "Nov" / "Dec"
day         = ([FWS] 1*2DIGIT) / obs-day
time        = time-of-day FWS zone
time-of-day = hour ":" minute [ ":" second ]
hour        = 2DIGIT / obs-hour
minute      = 2DIGIT / obs-minute
second      = 2DIGIT / obs-second
zone        = (( "+" / "-" ) 4DIGIT) / obs-zone
```

http://tools.ietf.org/html/rfc2822#section-3.3

FWS = Folding Whitespace

Array Objects

- A built-in ordered list-like collection object for holding a sequence of values
- Elements are accessed using `[]`, read and write
- Arrays are not typed – they may hold a mixture of data types
- Arrays may be created using a literal notation or a constructor

```
// Literal notation
var arr1 = [1, 'second', false];

// Constructor w/ initialization
var arr2 = new Array(1, 'second', false);

// Constructor w/ undefined values
var arr3 = new Array(3);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Array Object Methods

- `concat(…)`
- `fill(…)`
- `filter(…)`
- `find(…)`
- `forEach(…)`
- `indexOf(…)`
- `join(…)`
- `sort(…)`
- `reverse(…)`
- `pop(…)`
- `push(…)`
- `shift(…)`
- `unshift(…)`
- `slice(…)`
- `splice(…)`
- `map(…)`
- `reduce(…)`
- …

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Array Object Methods

C:\TCNJ\2016Spring\CSC470\Lectures\02\arrays.js

File Edit View Execute Environment Help

Open File... Save Save As... Run Inspect Display Pretty Print

```
// arrays.js

var arr1 = [1, 'second', false];          // Literal notation
var arr2 = new Array('A', true, null);    // Constructor with initialization
var arr3 = new Array(3);                  // Constructor with undefined values

console.log( arr1 );                      // Print an array
console.log( arr1.length )                // Print array length

arr1.push( 1.23 );                        // One way to add item to end
arr1[ arr1.length ] = 'last';             // Another way to add to end
console.log( arr1 );                      // Print new array and length
console.log( arr1.length )

console.log( arr1[ 0 ] );                 // Print first and last items
console.log( arr1[ arr1.length-1 ] );

var lastItem = arr1.pop();                // Remove last item
console.log(arr1);

arr1.unshift(0);                          // Add to front of list
console.log(arr1);

arr1.shift();                             // Remove from front of list
console.log(arr1);

var arr4 = arr1.concat( arr2 );           // Concatenate arrays
console.log(arr4);
```

Line 29, Col 1

Console - about:newtab

Ins... C... Deb... Style ... Perfor... Ne...

Net CSS JS Security Logging Server Clear

```
Array [ 1, "second", false ]                                  Scratchpad
3                                                             Scratchpad
Array [ 1, "second", false, 1.23, "last" ]                    Scratchpad/
5                                                             Scratchpad/
1                                                             Scratchpad/
last                                                          Scratchpad/
Array [ 1, "second", false, 1.23 ]                            Scratchpad/
Array [ 0, 1, "second", false, 1.23 ]                         Scratchpad/
Array [ 1, "second", false, 1.23 ]                            Scratchpad/
Array [ 1, "second", false, 1.23, "A", true, null ]           Scratchpad/
```

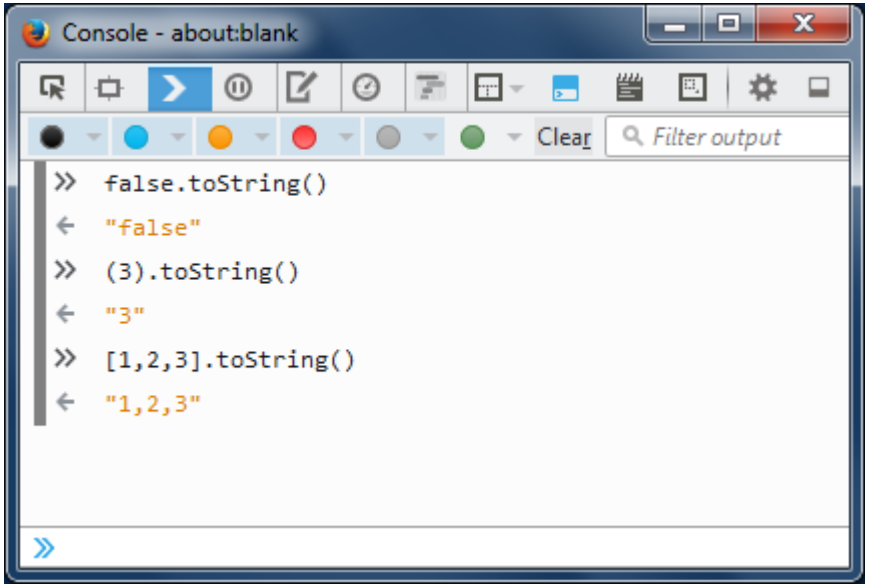
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Array Objects

- Arrays may have "holes"
- Empty slots are `undefined`
- Array items may be deleted with `delete`
- Deleting a item does not cause array items to shift down
- Arrays are like objects with numeric keys

Object

- A collection of properties where each property has a name and a value
  - Property names are strings, although they do not need to be quoted
  - Property values can be anything, including Arrays, Functions or other Objects
- A JavaScript Object can be thought of as an associative array (hash)
  - a.k.a. HashMap in Java, Dictionary in Python, Hash in Ruby, Hash Table in C++, Associative Array in PHP
- Nearly all things in JavaScript are objects. All objects inherit from Object and its methods, such as `toString()`



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object

Object Creation

Object Constructor

```
var ob = new Object();
```

Objects have literal notation:

- Object literal delineated by curly brackets
- Properties in literal are separated by commas (,)
- Property name:value pairs separated by a colon (:)

```
var character = {firstName: 'Bart', lastName: 'Simpson', age: 10,
                 school: 'Springfield Elementary', grade: 4};
```

Empty objects created using different methods are semantically equivalent

```
var ob1 = new Object();
var ob2 = {};
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object

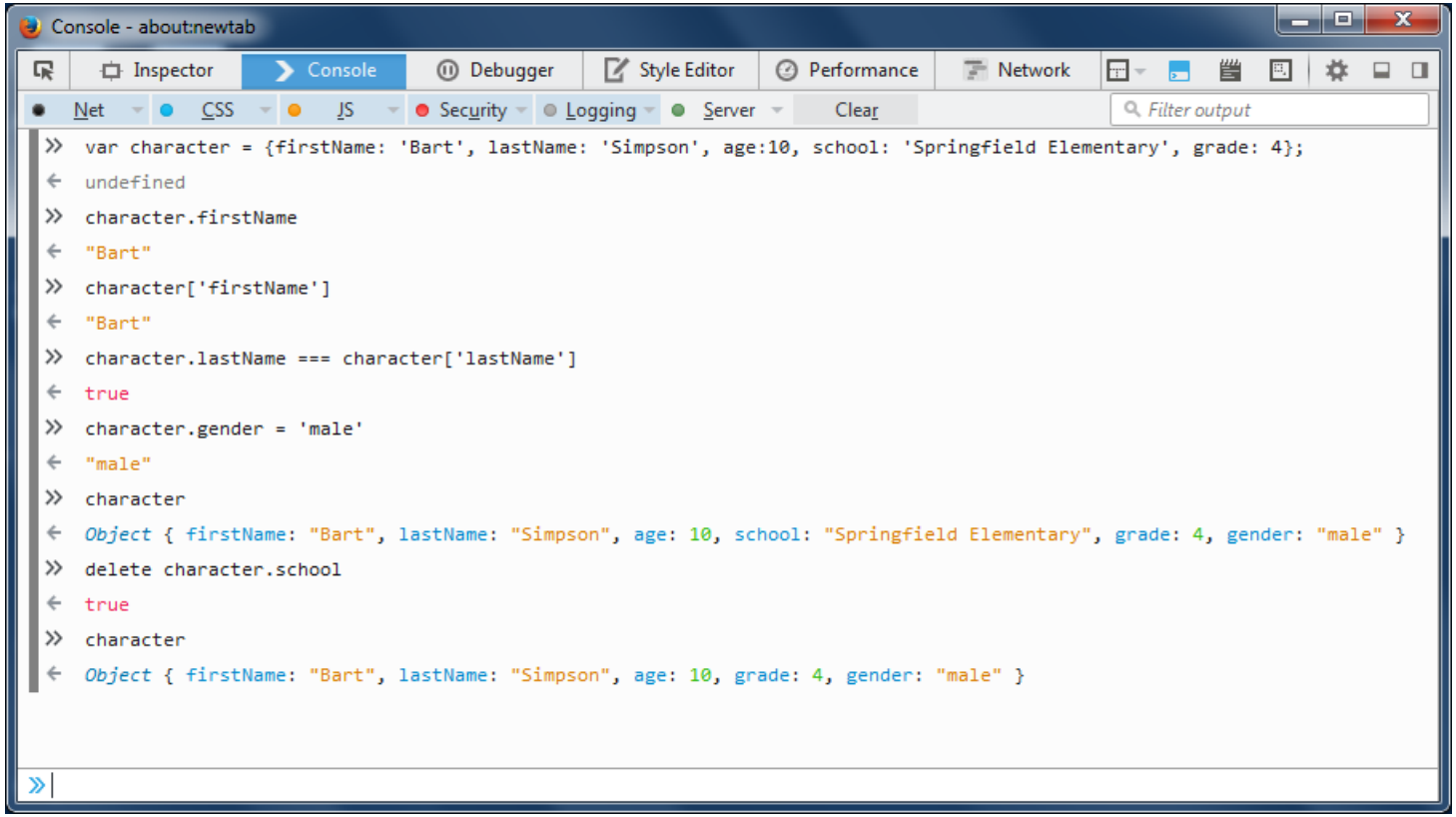
Object

Object properties may be accessed in <u>two ways</u>:

- Using dot-notation `character.firstName`
- Using square brackets `character['firstName']`

Object properties may be deleted using the `delete` command

- `delete character.school`



Objects

Object property values may be any type, including arrays and other objects

```
var character = {
    firstName: 'Bart',
    lastName: 'Simpson',
    age: 10,
    school: 'Springfield Elementary',
    grade: 4,
    family: [
        { firstName: 'Lisa',
          lastName: 'Simpson',
          age: 8,
          school: 'Springfield Elementary',
          grade: 2 },
        { firstName: Maggie',
          lastName: 'Simpson',
          age: 1,
          school: null,
          grade: null }
    ]
};
```

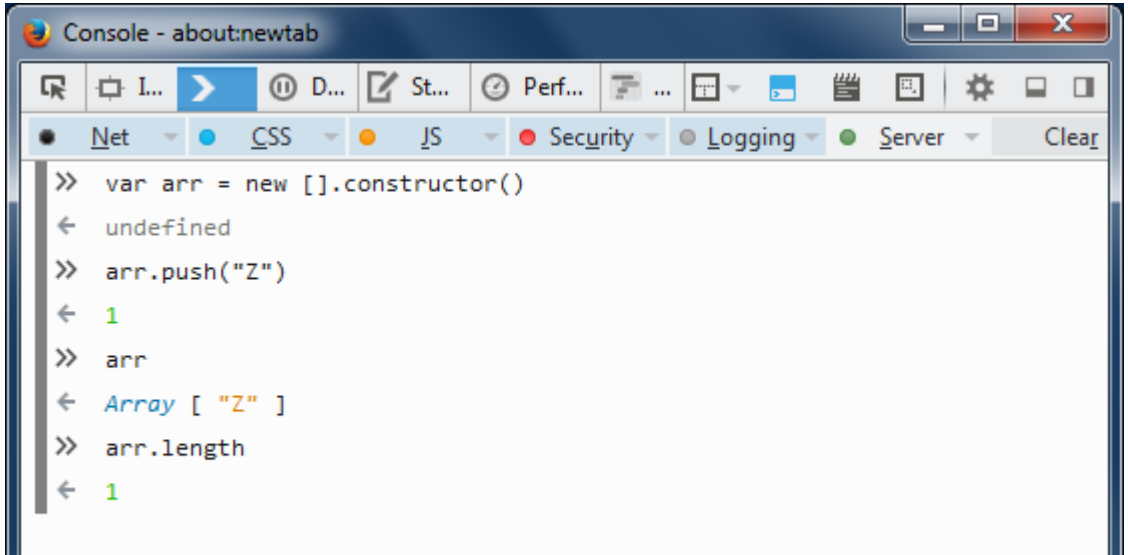
https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

Constructor Methods

Object constructor functions are available as the `constructor` property of an object.

- `"ABC".constructor`
- `(12).constructor`
- `(true).constructor`
- `[].constructor`
- `({}).constructor`

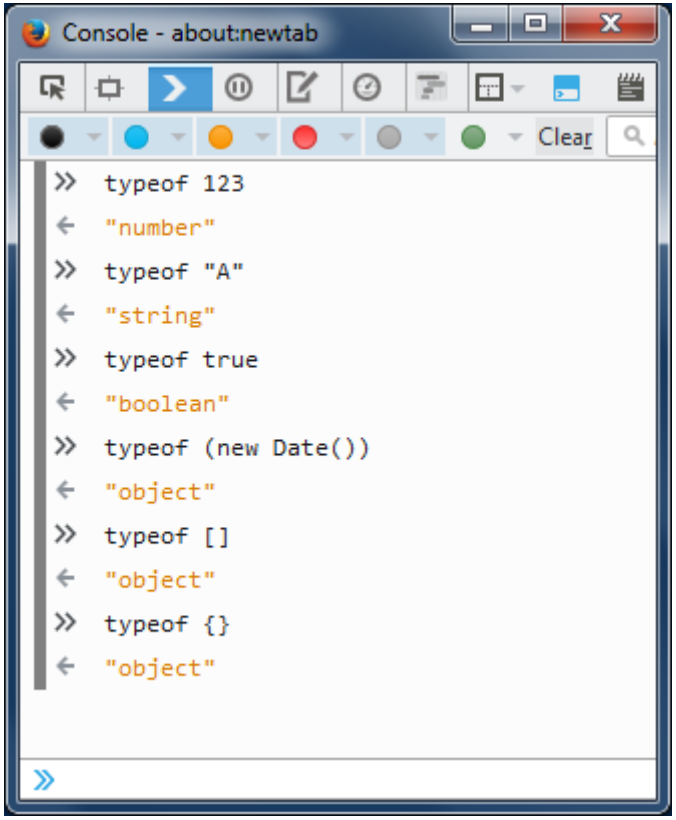
May be used to create new instances of an object



`typeof` Operator

(Recall)

A prefix operator that returns the object type as a `String`



`instanceof` Operator

- `instanceof` is a binary infix operator used to test the type of an object
- Gives expected result only when object is created with constructor
- By contrast, `typeof` operator produces expected result only for literals – values created with a constructor results in "object"
- Alternate method to test the `instanceof` a value is to compare the `constructor` method

C:\TCNJ\2016Spring\CSC470\Lectures\02\instanceof.js

File Edit View Execute Environment Help

Open File... Save Save As... Run Inspect Display

```
// instanceof.js

// Create several items
var str1 = new String("ABC");
var str2 = "ABC";
var num1 = new Number(2);
var num2 = 2;
var bln1 = new Boolean(true);
var bln2 = true;

// instanceof does not work with objects created
// using literal notation
console.log( str1 instanceof String );        // true
console.log( str2 instanceof String );        // false
console.log( num1 instanceof Number );        // true
console.log( num2 instanceof Number );        // false
console.log( bln1 instanceof Boolean );       // true
console.log( bln2 instanceof Boolean );       // false

// typeof does not work. The type of a String object is 'object'
console.log( (typeof str1) === "string" );    // false
console.log( (typeof str2) === "string" );    // true
console.log( (typeof num1) === "number" );    // false
console.log( (typeof num2) === "number" );    // true
console.log( (typeof bln1) === "boolean" );   // false
console.log( (typeof bln2) === "boolean" );   // true

// Alternate way to test the instanceof an object
console.log( str1.constructor === String );   // true
console.log( str2.constructor === String );   // true
console.log( num1.constructor === Number );   // true
console.log( num2.constructor === Number );   // true
console.log( bln1.constructor === Boolean );  // true
console.log( bln2.constructor === Boolean );  // true
```

Line 35, Col 1

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

The Global Object

- In JavaScript there is always a global object defined
  - In a web browser the `window` object is the global object
  - In Node.js the global object is `global`
- All global variables belong to the global object
- One way to get the global object is to evaluate "`this`" in the global scope

```
>> this === window
true
```

- Properties of this object are the globally defined symbols that are available to a JavaScript program

```
>> var name = 'Bart';
>> window.name
"Bart"
```

- When the JavaScript interpreter starts it creates a new global object and gives it an initial set of properties