



CSCI 165

Introduction to the Internet and the World Wide Web

Lec 0: Introduction, again



Jetic Gū

Overview

- Focus: Web Development
- Architecture: Internet
- Core Ideas:
 1. Administration
 2. Javascript

Administration

Course Arrangement

- Week 8-13: Jetic
 - Instructor @Columbia College, PhD Candidate in Computational Neuroscience
- Zoom Link
 - Well, you are already here
 - Lectures: the sooner we finish, the sooner we finish
- Course Website
 - <https://jetic.org/kurs/csci-165/>

Grading Scheme

- 5 Labs, released on Wednesday
- 1 Assignment
- Small Test
- Final Exam
 - About tests and exams, I **DO NOT** require you to turn on your **camera**. Most likely I will be working on **stuff** when you take the test, but I will check your code to see if it is copied.
- Project?

P1
Q?

Questions?

Summary

Javascript Continued

Basic Syntax, Sequential Execution, etc.

Variables

```
var name = expression;  
var clientName = "Connie Client";  
var age = 32;  
var weight = 127.4;
```

- variables are declared with the `var` keyword (case sensitive)
- types are not specified, but JS does have types ("loosely typed")
 - `Number`, `Boolean`, `String`, `Array`, `Object`, `Function`, `Null`, `Undefined`
- can find out a variable's type by calling `typeof`

Number type

```
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, Python, C++
- many operators auto-convert types: "2" * 3 is 6

Comments

(same as Java/C/C++)

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

- identical to Java and C++'s comment syntax
- recall: 4 comment syntaxes
 - HTML: `<!-- comment -->`
 - CSS/JS/PHP: `/* comment */`
 - Java/JS/PHP: `// comment`
 - PHP: `# comment`

Math object

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

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

Special values:

`null` and `undefined`

```
var ned = null;
var benson = 9;
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```

- `undefined` : has not been declared, does not exist
- `null` : exists, but was specifically assigned an empty or null value
- Why does JavaScript have both of these?

Logical operators

- Similar to Java/C/C++
- `>` `<` `>=` `<=` `&&` `||` `!` `==` `!=` `===` `!==`
- most logical operators automatically convert types:
 - `5 < "7"` is true
 - `42 == 42.0` is true
 - `"5.0" == 5` is true
- `===` and `!==` are strict equality tests; checks both type and value
 - `"5.0" === 5` is false

if/else statement (same as Java/C/C++)

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

- identical structure to Java's if/else statement
- JavaScript allows almost anything as a condition

Boolean type (same as Java/C/C++)

```
var iLike190M = true;  
var ieIsGood = "IE6" > 0; // false  
if ("web devevelopment is great") { /* true */ }  
if (0) { /* false */ }
```

- any value can be used as a Boolean
- "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;
```

for loop (same as Java/C/C++)

```
var sum = 0;
for (var i = 0; i < 100; i++) {
    sum = sum + i;
}

var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
    s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheelllloo"
```


while loops (same as Java/C/C++)

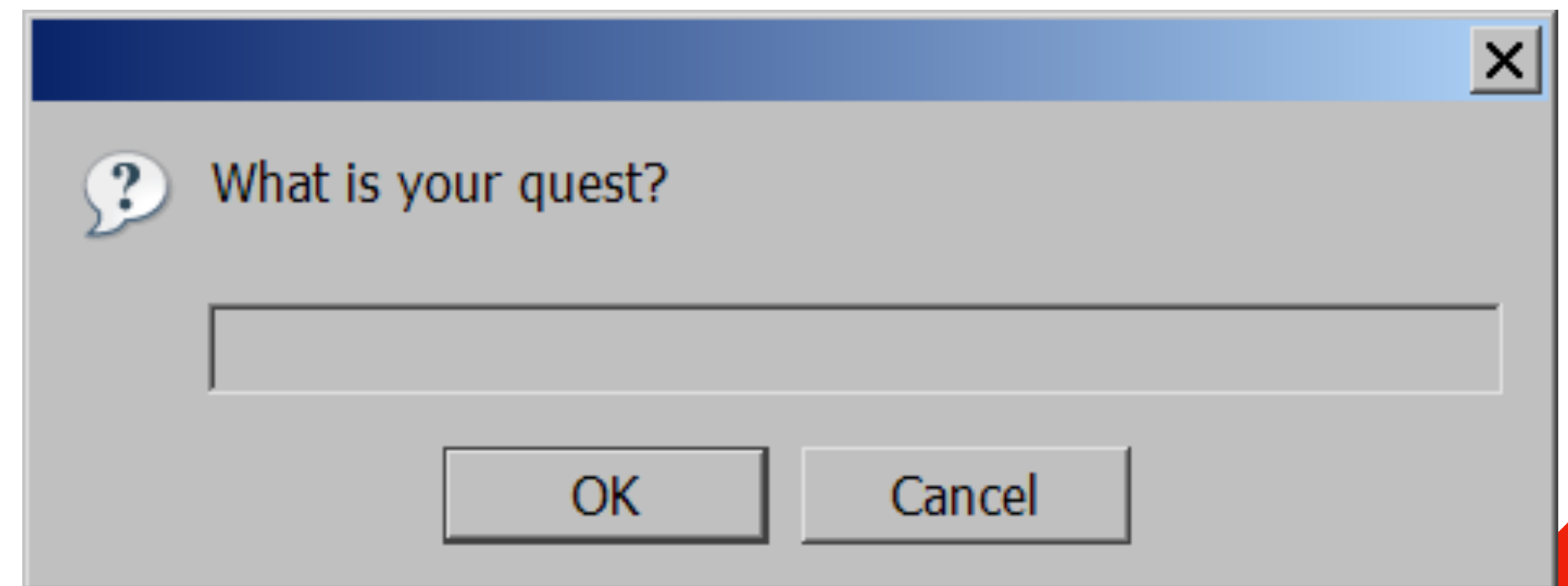
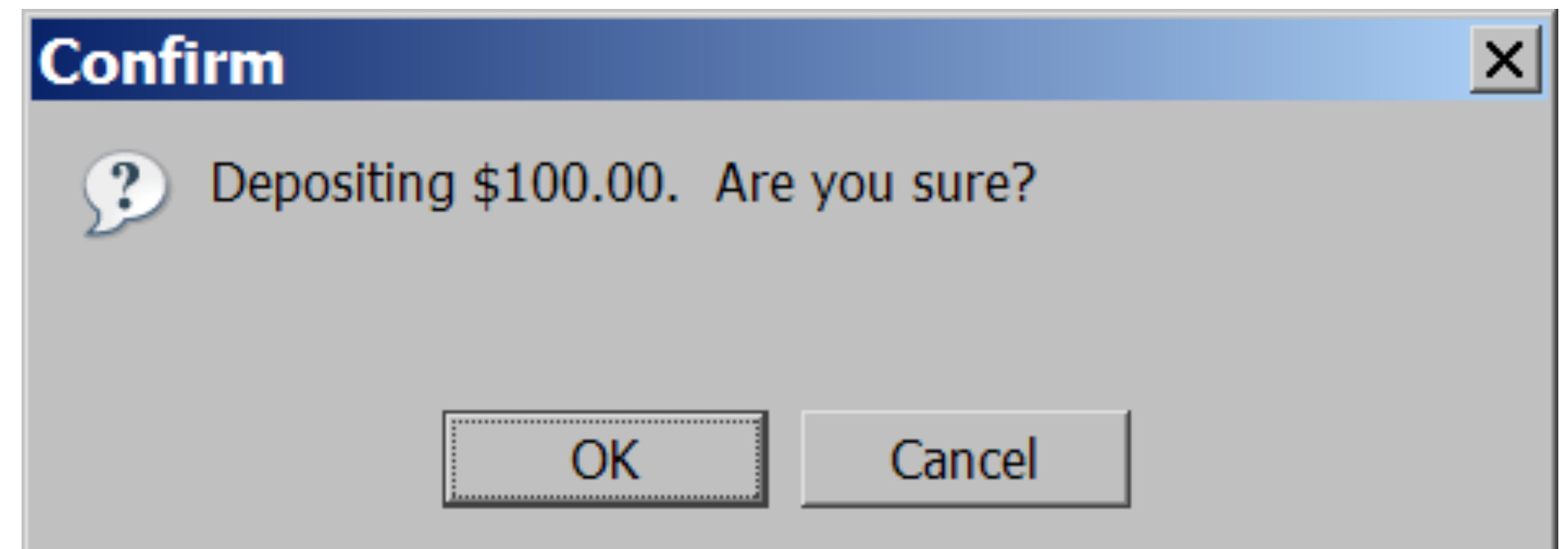
```
while (condition) {  
    statements;  
}
```

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

- break and continue keywords also behave as in Java/C/C++

Popup boxes

```
alert("message"); // message  
confirm("message"); // returns true or  
false  
prompt("message"); // returns user  
input string
```



Arrays

```
var name = []; // empty array  
var name = [value, value, ..., value]; // pre-filled  
name[index] = value; // store element
```

```
var ducks = ["Huey", "Dewey", "Louie"];  
var stooges = []; // stooges.length is 0  
stooges[0] = "Larry"; // stooges.length is 1  
stooges[1] = "Moe"; // stooges.length is 2  
stooges[4] = "Curly"; // stooges.length is 5  
stooges[4] = "Shemp"; // stooges.length is 5
```

Array methods

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

- array serves as many data structures: list, queue, stack, ...
- **methods:** concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
- push and pop add / remove from back
- unshift and shift add / remove from front
- shift and pop return the element that is removed

String type

- **methods:** `charAt`, `charCodeAt`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`, `length`
 - `charAt` returns a one-letter String (there is no char type)
- Strings can be specified with `""` or `''`
- concatenation with `+` :
 - `1 + 1` is `2`, but `"1" + 1` is `"11"`

```
var s = "Connie Client";  
var fName = s.substring(0, s.indexOf(" ")); // "Connie"  
var len = s.length; // 13  
var s2 = 'Melvin Merchant';
```

More about String

- escape sequences behave as in Java: `\ ' \ " \ & \ n \ t \ \`
- converting between numbers and Strings:

```
var count = 10;
var s1 = "" + count; // "10"
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah ah ah!"
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("booyah"); // NaN
```
- accessing the letters of a String:

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);
```

Splitting strings: split and join

```
var s = "the quick brown fox";  
var a = s.split(" "); // ["the", "quick", "brown", "fox"]  
a.reverse(); // ["fox", "brown", "quick", "the"]  
s = a.join("!"); // "fox!brown!quick!the"
```

- split breaks apart a string into an array using a delimiter
- can also be used with regular expressions
- join merges an array into a single string, placing a delimiter between them