Client Scripting
Advanced Functions
JavaScript Classes

• Work as a table
• For each person:
  – Explain your class
  – Explain your method extension of the Date, Math, Number, or String class
Method Chaining

• If a method does not otherwise return anything, then it can be useful for it to return *this*.
  – What is *this*?

• In this way, the returned value can be used to *chain* additional methods.
class Calculator {
    constructor (initValue) {
        this.value = initValue;
    }
    add (addend) {
        this.value += addend;
        return this;
    }
    sub (subtrahend) {
        this.value -= subtrahend;
        return this;
    }
    equals () {
        return this.value.toString();
    }
}

• You can then chain:
  var c = new Calculator(0);
  c.add(5).add(17).sub(8).equals();

• Method chaining is used a lot in jQuery.
  This code is at: http://www.andrew.cmu.edu/course/67-328/examples/calculator.html
Method Chaining Exercise

• Working in pairs
• Create a class with 3 methods that can chain.
• OK to use one of your solutions to today's homework
• Demonstrate it with an example.
Asynchronous Programming

- Browsers, mobile devices, and servers spend a lot of their time waiting for things to happen.
  - Waiting for a user to click a button
  - Waiting for a server to make a response
  - Waiting for a client to make a request
  - Waiting for a database to return data
Synchronous vs Asynchronous

• There are two programming styles for handling things that take time and you have to wait for:
  – Synchronous:
    • When initiating something that takes time, "block"
      – Stop and wait until the response is received
    • E.g
      – resultSet = getFromDatabase(query)
        <- stop until this request completes
      – use resultSet
  – Asynchronous:
    • When initiating something that takes time,
      – start it and provide a "handler" to run when the response is received
      – move on and do other things
    • E.g.
      – startGetFromDatabase(functionToCallUponCompletion)
      – go do other things
      – When the database operation is completed, functionToCallUponCompletion is called with the results
Asynchronous Programming

• Real-life metaphors for *asynchronously*
  – Events
    • My daughter's goofy dog: Barks wildly when she hears a doorbell
      – Even if doorbell is on the TV
      – Even if she is outside the door and her walker rang the bell
    • Otherwise she can deal with other tasks, like sleeping or eating.
    • She implements a listener function: on hearing doorbell (action), bark wildly
  – Timer
    • Set an alarm clock to wake up at 8:00am
    • A timer is a type of event, but programmatically generated.
    • You implement a listener function: upon alarm, wake up
  – Asynchronous processes
    • In order to have a pizza delivered:
      – call and order the pizza
      – give instructions for how to ring your apartment once they arrive at the building
    • These instructions are like a callback function, a function you give the process (pizza person) to do when the task is completed
    • Between ordering pizza and answering door, you can do other things
Asynchronous Programming

• Client-side (browser) **asynchronosity**
  – User Events
    • E.g. onclick, onmouseover, onfocus
    • And lots more
  – Timer
    • setTimeout - do callback after some given time
      – This is a built-in JavaScript function
      – setTimeout(functionToCall, numMilliseconds)
    • setInterval – do callback after every given time interval
      – Also a built-in JavaScript function
      – setInterval(functionToCall, numMilliseconds)
  – Asynchronous processes
    • E.g. make a request to the server (AJAX) and run a callback function when the reply is completed
function tick() {
    console.log("tick ");
    setTimeout(tock, 1000);
}

function tock() {
    console.log("tock ");
    setTimeout(tick, 1000);
}

Callback function is passed to setTimeout
"When you hit 1000 ms, call this function."

This code is at: http://www.andrew.cmu.edu/course/67-328/examples/tick.html
Anonymous function as a callback

- Anonymous functions are often used as a callback
- I.e. functions defined in-line, without names
- This version works identically to the last one

Or using ES6 arrow functions

```javascript
function tick() {
  console.log("tick ");
  setTimeout(function(){
    console.log("tock ");
    setTimeout(tick, 1000);
  }, 1000);
}

function tick() {
  console.log("tick ");
  setTimeout(() => {
    console.log("tock ");
    setTimeout(tick, 1000);
  }, 1000);
}
```

This code is at: http://www.andrew.cmu.edu/course/67-328/examples/tick2.html
• **Without using a global variable,** have tick and tock produce the following:
  
tick 0
  tock 1
  tick 2
  tock 3
  tick 4
  tock 5
  ...

© Joe Mertz - Distributed Application Development
Would the following work?

```javascript
function tick() {
    let tcount = 0;
    console.log("tick "+ tcount++);
    setTimeout(function() {
        console.log("tock "+ tcount++);
        setTimeout(tick, 1000);
        }, 1000);
    }
}
```

This code is at: http://www.andrew.cmu.edu/course/67-328/examples/tick3a.html
Closure exercise

• Thinking it through...
  – You need a variable that is in scope for tick and tock
  – But it cannot be in the global scope
  – Therefore you need a variable in the scope of both tick and tock
  – This implies that tick and tock must be nested within a 3rd function (e.g. startClock).
  – The tick and tock counter (e.g. tcount) can be a variable in this outer function, and by closure it will be within scope of tick and tock (even after the outer function has completed executing).

• Continue working on the exercise...
One solution (2 global variables)

tickTock = function() {
    let tcount = 0;
    function tick() {
        console.log("tick "+ tcount++);
        setTimeout(tock, 1000);
    }
    function tock() {
        console.log("tock "+ tcount++);
        setTimeout(tick, 1000);
    }
    return tick;   // tickTock is a function that returns a function
};

// run tickTock to get the tick function
startTickTock = tickTock();

// startTickTock starts the process of ticking and tocking
startTickTock();
var startClock = (function() {
    let tcount = 0;
    function tick() {
        console.log("tick "+ tcount++);
        setTimeout(tock, 1000);
    }
    function tock() {
        console.log("tock "+ tcount++);
        setTimeout(tick, 1000);
    }
    return tick;
})(());

This code is at: http://www.andrew.cmu.edu/course/67-328/examples/tick3b.html
Zero global variables

(function() {
    var tcount = 0;
    function tick() {
        console.log("tick "+ tcount++);
        setTimeout(tock, 1000);
    }
    function tock() {
        console.log("tock "+ tcount++);
        setTimeout(tick, 1000);
    }
    return tick;
})();()();