1. Announcements
   a. Details about midterm
      i. Open book, open notes, calculators ok, no Google,
   b. Review of some problems, maybe
   c. Review of the review session
2. Programming in a language
   a. Key distinction
      i. Syntax
         1. Grammar rules for defining legal statements
      ii. Semantics
         1. What statements actually mean (how they compute)
   b. Why Javascript?
      i. Can use it in web pages
         1. Easy to see what the program does
         2. Easy to hand in assignment, ...
      ii. Easy language to start with
         1. Can do useful things fairly quickly
      iii. Ideas carry over to other languages
      iv. Has limitations
         1. Some irregularities and surprising behaviors
         2. Different browsers do things differently; none completely fits with specification
         3. Would never use it to write large programs
   c. What we’re going to do
      i. Lab 5 – a guessing game; similar to binary search exercise done in class
      ii. Lab 6 – an arcade game; whack-a-mole (hit the object a few times to win)
      iii. Some problem set questions about semantics
   d. Learning a programming language
      i. Language is less important than problem solving
         1. Think about how you would solve
            a. May involve building a flow chart
         2. Work out the steps
         3. Then translate into language
      ii. Language syntax is (obviously) important
         1. But can look it up
         2. No need to memorize
      iii. Libraries and built-in functions
         1. Not necessary to remember syntax
         2. Worthwhile to have a good sense of what you can do easily
   e. Big picture for Javascript
      i. Code appears in an html file between <script> and </script> tags
      ii. So
         1. <script>
         2. Javascript code
3. Remember what you need for a programming language
   a. Ability to define variables and constants
      i. Can be strings, integers, floating point numbers
      ii. To define
         1. var x, y, z;
         2. var x = 0;
      iii. In place of having to declare each variable separately
   b. Operations
      i. Usual +, -, *, /
      ii. Also string operations
         1. ‘hello ‘ + “world!” = ‘hello world’
         2. But, ‘hello’ + ‘world’ = ‘hellowworld’
      iii. Because of this, have to differentiate between strings and numbers
   c. Branching and looping
      i. Show operations
      ii. More please than IFZERO, IFPOS, GOTO
   d. Also, built in operations and user defined functions
      i. E.g to read input and write output
         1. Prompt() does what GET did
         2. Alert() does what PRINT did

4. First simple example
   a. Version of Toy Program
      i. GET
      ii. PRINT
      iii. STOP
   b. Javascript
      <html>
      <body>
      <p> name2.html: joins 2 names
      <script>
      var firstname, secondname, result;
      firstname = prompt("Enter first name");
      secondname = prompt("Enter last name");
      result = firstname + " " +secondname;  // + means "join" here
      alert("hello, " + result);  // and here
      </script>
      
5. Next example – various versions to illustrate differences between strings and numbers
   a. First version
      <script>
      var a, b;
      a = 1;
b = 2;
alert(a+b); // prints 3

b. Next version

```html
<script>
var a,b;
a =’1’;
b = ’2’;
alert(a+b); // prints ’12’
</script>
```

c. Third version (what does this do?)

```html
<script>
var a,b;
a = prompt(); // enter 1
b = prompt(); // enter 2
alert(a+b); // prints 3
</script>
```

d. To fix things if you wanted numbers, change program to

```html
<script>
var a,b;
a = prompt();
a = parseInt(a); // could also write a = parseInt(prompt());
b = prompt();
b = parseInt(b);
alert(a+b); // prints 3
</script>
```

e. parseInt() takes a string to an integer; assumption is that input is a string and not an integer

f. parseFloat() takes a string to a floating point number

g. Need to be careful to be sure that Javascript knows your intention

6. Next, testing and looping

a. Testing in Javascript

i. Comparison operators

1. `==` tests for equality
   a. 1 == 1, “1” == 1, 1 == true are all true (equal to 1)
   b. 1 == false, 1 == 2 are false (equal to 0)

2. `!=` tests for inequality

3. `>` (greater than), `>=` (greater than or equal to), `<` (less than), `<=` (less than or equal to), ... can compare numerical quantities

b. Looping in Javascript

i. Simple loop

```javascript
while (condition) {
    Block of instructions to be done
}
// end of loop
```
ii. Simple branch
   if (condition) {
       Block of instructions to be executed
   }

iii. More complex branch
   if (condition) {
       Block of instructions to be executed
   } else {
       Alternative block of instructions to be executed
   }

iv. Can embed further
   1. If (...){ ... } else if ( ... ) { ... } else { ... } // and further

7. Example program to add numbers until input is 0
   <html>
   <body>
   <script>
   var sum = 0;
   var num;
   num = prompt("Enter new value, or 0 to end");
   while (num != 0) {
       sum = sum + parseInt(num);
       num = prompt("Enter new value, or 0 to end");
   }
   alert("Sum = " + sum);
   </script>
   </body>
   </html>

8. Example program to find the largest number
   var max = 0;
   var num;
   num = prompt("Enter new value, or 0 to end");
   while (num != 0) {
       if (parseFloat(num) > max)
           max = num;
       num = prompt("Enter new value, or 0 to end");
   }
   document.write("<P> Max = " + max); // writes to html; doesn’t appear until done

9. Example program using operators and looping and looking further into prompt()
   a. Prompt() either brings back a value (if OK button is hit after entering input) or null (if cancel button is clicked)
      i. Allows 0 to be a possible input
   b. Example program
      var rad, area;
      rad = prompt("Enter radius");
      while (rad !== null) {
          area = 3.14 * rad * rad;
document.write("<p> radius = " + rad + ", area = " + area);
    
    rad = prompt("Enter radius");
}

c. Another example – embedded tests; determine if number is positive, negative or zero
    var num = prompt("Enter number");
    while (num != null) {
        num = parseInt(num);
        if (num > 0) {
            alert(num + " is positive");
        } else if (num < 0) {
            alert(num + " is negative");
        } else {
            alert(num + " is zero");
        }
        num = prompt("Enter number");
    }

10. Creepy or not
    a. Riley v. California -- can police search a cell phone
    c. Cell phones differ in both a quantitative and a qualitativesense from other objects that might
be carried on an arrestee’s person. Notably, modern cell phones have an immense storage
capacity. Before cell phones, a search of a person was limited by physical realities and
generally constituted only a narrow intrusion on privacy. But cell phones can store millions of
pages of text, thousands of pictures, or hundreds of videos. This has several interrelated
privacy consequences. First, a cell phone collects in one place many distincttypes of
information that reveal much more in combination than anyisolated record. Second, the
phone’s capacity allows even just one type of information to convey far more than previously
possible. Third, data on the phone can date back for years. In addition, an element of
pervasiveness characterizes cell phones but not physicalrecords. A decade ago officers might
have occasionally stumbledacross a highly personal item such as a diary, but today many of
the more than 90% of American adults who own cell phones keep on theirperson a digital
record of nearly every aspect of their lives.

11. Functions
    a. If you are going to do something many times, write a function
    b. What is a function
    i. Statements are collected in one place and given a name
    ii. Other parts of the program can “call” the function
    1. That is, use the function without copying it
    iii. Functions can take input values
    c. Functions we’ve already seen
    i. prompt()
    ii. alert()
    d. We can write our own function
    i. Syntax for doing so
    
        function name_of_function(list of arguments)  {
            The statements of the function
ii. Example function

    function add_two_numbers(a,b) {
        // adds 2 numbers represented as strings
        return parseInt(a) + parseInt(b);
    }

iii. Usage

    var a,b;
    a = prompt('Enter your first number');
    b = prompt('Enter your second number');
    alert('The sum of ' + a + ' and ' + b + ' is ' + add_two_numbers(a,b));

e. Can embed functions

    function add_two_numbers(a,b) {
        return parseInt(a) + parseInt(b);
    }

    var a, b, c, d;
    a = prompt('give me a number');
    b = prompt('second number');
    c = prompt('third number');
    d = add_two_numbers(add_two_numbers(a,b),c);
    alert('The sum of ' + a + ' and ' + b + ' and ' + c + ' is ' +
         add_two_numbers(add_two_numbers(a,b),c));

f. Value of functions

    i. Function breaks a problem down into smaller pieces
       1. Each function can be tested separately when debugging

    ii. For larger programs
       1. Defines interfaces
       2. Allows people to work on different parts of the project

12. Summary of the pieces of a programming language

    a. constants: literal values like 1, 3.14, "Error!"
    b. variables: places to store data and results during computing
    c. declarations: specify name (and type) of variables, etc.
    d. expressions: operations on variables and constants to produce new values
    e. assignment: store a new value in a variable
    f. statements: assignment, input/output, loop, conditional, call
    g. conditionals: compare and branch; if-else
    h. loops: repeat statements while a condition is true
    i. functions: package a group of statements so they can be called/used from other places in a program
    j. libraries: functions already written for you