ANNOUNCEMENTS

- No posting of code in the forum
- Check class announcements daily

INFINITE LOOPS

- An infinite loop occurs when the expression controlling the loop never becomes false
- **Example1**

```
var x = 30;
while(x  >  0) {
    document.writeln("<li>Element</li>");
}
```

- **Example2**

```
var  x = 7;  // how about x = 8
while (x != 0)  {
    document.writeln("<li>Element</li>");
    x = x – 2;
}
```

- How can we detect infinite loops?

PROGRAMMING ERRORS

- **Syntax Error**: (Compile-time error) The program violates the language's grammar
- **Semantic Error:** The program fails to accomplish what we want
- **Debugging:** The process of finding and fixing errors. Extremely hard for large software systems. Tools for debugging:
  - Trace tables
  - Output statements
  - Debuggers
- **Analogy:**
  - Taco tom ate.→ Syntactically therefore semantically incorrect.
  - A taco ate tom.→ Syntactically correct however semantically incorrect.
  - Tom ate a taco → Syntactically and semantically correct (what we want!)

HOW TO FIND PROBLEMS IN YOUR CODE

- The process of finding problems in computer code is called debugging
- Why the word debugging? See first computer bug at:
  - http://www.jamesshuggins.com/h/tek1/first_computer_bug_large.htm
- Computer programming is NOT about writing code and letting someone else find the problems (bugs) it may have
- You have to learn how to find problems in your code
- First approach: output statements
  - Using alert
- Advanced approach
  - Debugger software

SUGGESTIONS FOR WRITING PROGRAMS

- **Design** → Make sure you first come up with a plan/design for your code (e.g., using pseudocode)
- **Do not wait until the last minute** → Code implementation can be unpredictable
- **Incremental code development** → Fundamental principle in computer programming. Write a little bit of code, and make sure it works before you move forward
- **Don't make assumptions** → If you are not clear about a language construct write a little program to familiarize yourself with the construct
- **Good Indentation** → From the get-go use good indentation as it will allow you to understand your code better

SUGGESTIONS FOR WRITING PROGRAMS

- **Good variable names** → Use good variable names from the beginning (do not use x and y and then change them to meaningful names before submitting the project)
- **Testing**
  - Test your code with simple cases first
  - Test as your develop your code
- **Keep backups** → As you make significant progress in your development, make the appropriate backups
  - Use submit server as a backup mechanism
- **Trace your code**
- **Use a debugger**
- **Take breaks** →If you cannot find a bug take a break and come back later

DESIGNING USING PSEUDOCODE

- Pseudocode can be seen as algorithm, outline of what you need to do
- So far we have focused on the syntax and semantics of JavaScript
- As the complexity of problems increase you need a design strategy to solve such problems
- Several alternatives exist to come up with a solution to a problem. A popular one is Pseudocode

Pseudocode: *English-like description of the set of steps required to solve a problem*

- When you write pseudocode you focus on determining the steps necessary to solve a problem without worrying about programming language syntax issues

PSEUDOCODE EXAMPLE

Pseudocode for finding the minimum value

1. Read number of values to process (call this value n)
2. Repeat the following steps until the n input values have been processed
   a. Read next value into x
   b. If (x is the first value read) {
      currentMinimum = x
      } else {
      if (x < currentMinimum)
      currentMinimum = x
      }
3. Print currentMinimum value

PSEUDOCODE ELEMENTS

- When writing pseudocode you need the following constructs:
  - Input
  - Output
  - Assignments
  - Repetition Structures
  - Conditionals
- To help you with the design of pseudocode you can use the following syntax to represent the above constructs

PSEUDOCODE ELEMENTS

- **Input**

  variable = read() e.g., x = read()

- **Output**

  print(variable) e.g., print(x)

- **Assignment**

  x = <value> e.g., x = 20, s = “Bob”

- **Repetition**

```
while (expression) { OR    do {
    stmts                      stmts
}                          } while (expression)
```

- Notice the above constructs look like JavaScript code but they are not JavaScript code

PSEUDOCODE ELEMENTS

Conditional (1)

```
if (expression) {
    stmts
}
```

Conditional(2)

```
if (expression) {
    stmts
} else {
    stmts
}
```

Conditional (3)

```
if (expression1) {
    stmts
} else if (expression2) {
    stmts
    …
} else if (expressionN) {
    stmts
} else {
    stmts
}
```

- For comparisons use: ===, <, >, <=, >=
- Notice the above constructs look like JavaScript code but they are not JavaScript code

HOW GOOD IS YOUR PSEUDOCODE

- Your code does not use language constructs that are particular to a programming language

- Anyone receiving the pseudocode will not need to ask you questions in order to transform the pseudocode into code (no matter what the target programming language is)

REVIEWING DEBUGGING TOOLS

- Let’s review how to use Chrome’s console
- Let’s review how to use Cloure Compiler

IN-CLASS LAB (20 MINUTES)