Announcements

- No posting of code in the forum
- Check class announcements daily
- You must implement programming projects by yourself
JavaScript Reference

JavaScript (If Statement)

- What is the difference between `===` and `==`?
- Let’s compare
  - `20 === “20”` vs. `20 == “20”`
JavaScript (Logical Operators)

- Used with comparison operators to create more complex expressions

Operators

- Logical and (&&) → expr1 && expr2
  - Expression is true if and only if both expressions are true otherwise is false
  - You can have more than two expressions
  - **Example:** LogicalOp1.html

- Logical or (||) → expr1 || expr2
  - Expression is false if and only if both expressions are false otherwise is true
  - You can have more than two expressions
  - **Example:** LogicalOp2.html

- Logical Not (!) → !expr
  - Inverts the boolean value of the expression
Precedence/Associativity

- Remember you can use parenthesis to impose a particular order for the evaluation of an expression
Cascaded If Statement Idiom

- You can combine if statements to handle different cases
- This approach to organize if statements to handle different cases is called the **Cascaded If Statement**
- Cascaded If statement general form:

```java
If (expr1) {
    // Statements executed if expr1 is true
} else if (expr2) {
    // Statements executed if expr2 is true
} else if (expr3) {
    // Statements executed if expr3 is true
} else {
    // If none of the above expressions is true
}
```

- Notice it is not a JavaScript statement
- Once one of the cases is executed no other case will be executed
- You don’t need to use `{ }` if you only have one statement
- More efficient than having multiple if statements
- **Example:** See CascadedIf.html
while Statement

- **while statement** → Control statement which allows JavaScript to repeat a set of statements
- **Basic Form**
  
  ```javascript
  while (expression) {
      statement(s)  // executed as long as expression is true
  }
  ```
- `{ }` not needed if you only have one statement
- You can have other types of statements (including whiles) in a while
- Common mistake: to add a semicolon after closing parenthesis
- **Example**: Numbers.html
- **Example**: EvenNumbers.html
- **Example**: NumbersTable.html
- **Example**: SqrtTable.html
Trace Tables

- Mechanism to keep track of values in a program
- Allows you to understand the program behavior
- We could create a trace table for EvenNumbers.html
Combination of Statements

- Keep in mind that you can have any combination of conditionals, and iteration (while) statements
- For example:
  - Conditionals inside of loops
  - Conditionals inside conditionals
  - Loops inside conditionals
  - Loops inside of loops
Infinite Loops

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

**Example 1**

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

**Example 2**

```java
int 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.htm](http://www.jamesshuggins.com/h/tek1/first_computer_bug.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
JavaScript Debugging Resources

- Let’s go over the information available at:
  
  http://www.cs.umd.edu/~nelson/classes/utilities/JavaScriptDebugging/

- Bookmark the above link and whenever you have a JavaScript problem use it 😊

- The link is also available in the Resources section of the class web page.
Suggestions for Solving Problems Using a Programming Language

- **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 Solving Problems Using a Programming Language

- **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