JavaScript (Loosely Typed Language)

- JavaScript is a loosely typed language
  - You do no need to specify a type for a variable
  - A variable can assume different types of values (not all at the same time)
- Code Snippet where variable value first assumes a string value and then a numeric value

```
var value, input, ratePerHour = 8.0;
value = "University of Maryland College Park";
document.writeln("Check will be sent to: " + value + "<br>");
input = prompt("Enter number of hours:");
value = input * ratePerHour;
document.writeln("Total amount is: " + value);
```

JavaScript (Data Types)

- We have seen integer, float, and string values
- New type: boolean type
  - Assumes the value true or false
  - Variable declaration and initialization

    var found = true;

    var attending = false;
- Some mathematical functions and constants you can use while working with numbers
  - Math.abs() – Absolute value (Example: Math.abs(-10))
  - Math.max() – Maximum of two values (Example: Math.max(10, 20))
  - Math.sqrt() – Square root (Example: Math.sqrt(4))
  - Math.random() – Random value between 0 and 1.0 (Example: Math.random())
  - Math.PI – mathematical constant pi

JavaScript (Comparisons)

- You can compare values by using the following operators
  - != → Returns true if the values are different, false otherwise (Example: x != y)
  - == → Return true if the values are equal, false otherwise (Example: x == y)
  - Relational Operators
    - < → Less than Returns true if left value is less than right value (Example: x < y)
    - > → Greater than
    - <= → Less than or equal
    - >= → Greater than or equal

JavaScript (If Statement)

- If statement – Control statement which allows JavaScript to make decisions
- It has two forms
- **First Form**

  if (expression)

  statement // executed if expression is true

- **Second Form**

  if (expression)

  statement1 // executed if expression is true

  else

  statement2 // executed if expression is false

- If you want to execute more than one statement then use a set of { } to enclose the statements
- **Example1** (See Example1.html)

JavaScript (while statement)

- while statement – Control statement which allows JavaScript to repeat a set of statements
- **Basic Form**

  while (expression)

  statement // executed as long as expression is true

- If you want to execute more than one statement then use a set of { } to enclose the statements
- **Example2** (See Example2.html)

JavaScript (do while statement)

- do while statement – Control statement which allows JavaScript to repeat a set of statements.
- **Basic Form**

  do

  statement // execute as long as expression is true

  while (expression);

- Executes the statement at least once
- If you want to execute more than one statement then use a set of { } to enclose the statements
- **Example3** (See Example3.html)
- do while statements can be nested
- alert – Used to generate a dialog box

JavaScript (Combination of Statements)

- You can combine if, while and do while statements in several ways
  - You can have an several nested if statements
  - You can have if statements within while statements
  - You can have while statements within if
- **Example4** (See Example4.html)
  - Example of nested loops
- Infinite loop
  - Condition where the controlling expression in a loop never becomes false

Designing Solutions Using Pseudocode

- So far we have focus on the syntax and semantics of JavaScript construct
- As the complexity of problems increases you need a strategy for designing solutions to such problems.
- Several alternatives exist to come up with a solution to a problem (which we can then turned into a computer program)
  1. **Flowchart:** Graphical representation of the solution, where symbols represent input/output operations, conditionals, and iteration statements.
  2. **Pseudocode:** English-like description of the set of steps required to solve a problem. **This is the alternative we want to use.** It represents our design!
- When you write pseudocode you focus on determining the steps necessary to solve a problem without worrying about JavaScript language syntax issues.

Pseudocode Elements

- When writing pseudocode you need three fundamentals 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:
  - **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 {
}                                   while (expression)
```

  - **Conditional**

```
if (expression) {           OR      if (expression) {           OR      if (expression1) {
                                        statements
    statements
    statements                      } else {                            } else if
    (expression2) {
                                        statements
    statements
                                                                        } else {
}                                   }                                     statements
                                                                        }
```

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

Pseudocode Example

```
entries = read()
prev = read()
i = 1
valid = true
while (i < entries && valid) {
      curr = read()
      if (curr < prev) {
            valid = false;
      }
      prev = curr
      i = i + 1
}
print("Sequence is:")
print(valid)
```

- Run always a trace table to verify your pseudocode is correct
- Could you implement the above code without actually knowing what it does??

Suggestions for Implementation

- **Make sure you have written pseudocode**
- **Do not wait until the last minute –** Code implementation could 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

JavaScript (Logical Operators)

- Used with comparison operators to create more complex expressions
- Operators
  - Logical and (&&) – expr1 && expr2
    - The whole expression is true if and only if both expressions are true otherwise is false
  - Logical or (||) – expr1 || expr2
    - The whole expression is false if and only if both expressions are false otherwise is true
  - Logical Not (!) – !expr
    - Inverts the boolean value of the expression

JavaScript (Cascaded If Statement Idiom)

- You can combine if statements to handle different cases
- This approach to organized if statements in order to handle different cases is called the Cascaded If Statement
- Cascaded If statement general form

```
If (expr1)
      // Statement is executed if expr1 is true
else if (expr2)
      // Statement is executed if  expr2 is true
else if (expr3)
      // Statement is 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 can use { } to enclose more than one statement
- **Example5** (See Example5.html)

JavaScript (Functions)

- Function - An entity that completes a particular task for us. It can takes values necessary to complete the particular task and it can return values.
- Examples of JavaScript functions
  - document.writeln
  - Alert()
- You can define your own functions.
- Let's see an example of a function that computes a letter grade based on a numeric grade.
- **Example6** (See Example6.html)
- How is the function executed?
  - Arguments are passed to the function
  - The letter grade is computed
  - Result is returned via the **return** statement

JavaScript (Functions)

- General form of a function is:

  function name (<comma-separated list of parameters>) {
  statements
  }

- Functions are invoked by using the () operator after the function's name
- Some functions may not return a value
- Some functions may not take any parameters
- There are other approaches to define functions besides the above approach
- Some advantages of functions are
  - Allow you factor out common code
  - Allow you to reuse code