**Lesson 13 – Recursion**

* Recursion
  + An Algorithm where a method calls itself as part of the solution to a problem
    - Can solve problems that are very difficult to solve with iterative solutions
  + A form of looping which a method calls itself to solve simpler versions of the problem
* Mathematical example of Recursion
  + 1! = 1
  + 2! = 2\*1 or 2\*1!
    - Each line can be solved in terms of the previous line
* Recursion Properties
  + Must solve the problem by breaking it down into smaller simpler problems of the same type
  + Must contain a base case that is eventually reached for each recursive call
  + Is a repetition structure which usually contains if statements rather than conventional loops
* Recursion Process
  + Recursion involves the internal use of a **stack**
    - **Stack** is a data abstraction where:
      * New data is “pushed” or added to the top of the stack
      * When information is removed from the stack it is “popped” or removed from the top of the stack
  + The recursive calls of a method will be stored on a stack and manipulated in a similar manner
  + The current computation is temporarily suspended and placed on the stack with all of its current information available for later use
    - Suspended in memory
  + A completely new copy of the method is used to evaluate the recursive call
  + When the base case is encountered the recursion will now unwind and result in a final answer
  + The value returned by the recursive call is used to complete the suspended computation
* Recursive Errors
  + If the recursion never reaches the base case the recursive calls will continue until the computer runs out of memory and the program crashes
    - May result in a StackOverFlow or Heap Storage Exhaustion error
      * Prevent this by ensuring it reaches the base case