CS 125 - Lecture 31

1. **Recursion: allows us to define a function that calls itself to solve a problem by breaking it into simpler cases.**

Learning recursion

Static: definitions. In code: magic!

Dynamic: taking the magic out.

2a. **Iteratively**

```
public class Example001 {

   public static void countdown(int n) {
      for (int i=n; i > 0; i--) {
         System.out.println(i);
      }
      System.out.println("Blast Off!");
   }

   public static void main(String[] args) {
      countdown(5);
   }
}
```

2b. **Recursively**

```
public class Example002 {

   public static void countdown(int n) {
      if (n == 0) {
         System.out.println("Blast Off!");
      } else {
         System.out.println(n);
         countdown(n-1);
      }
   }

   public static void main(String[] args) {
      countdown(5);
   }
}
```

Recursive algorithms composed of two cases:

1) **recursive case** calls the recursive procedure on a simpler case (usually a part of the input).

2) **base case** is <u>necessary</u> in recursion; it determines when the procedure returns a value (or terminates), rather than continuing the recursive process.

3. x^y

Raising a number to a power in java? x^y ???

No! Write two versions: iterative, recursive:

$$x^y = \underbrace{x * x * x * . . . x}_{\text{y times}}$$

3a. **Iteratively**

```
public class MyMath {

 public static int power(int base, int exp)
  {

  }

   public static void main(String[] args) {
      System.out.println(power(10,3));
   }
}
```

3b. **Recursively**

```
public class MyMath {

public static int power(int base, int exp)
{

 }

 public static void main(String[] args) {
   System.out.println(power(10,3));
 }
}
```

4a. **Write a java class to create a linked list.**

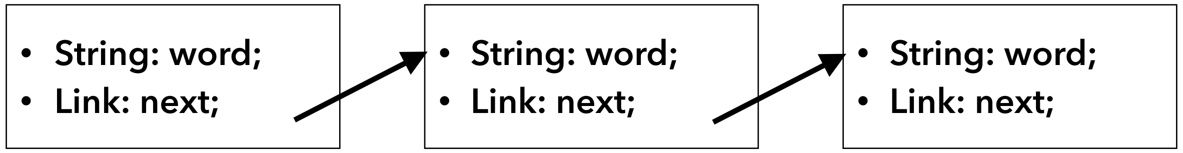
Write a java class to create a linked list.
Each Link object contains: i) an String ‘word’
ii) a reference 'next' to refer to the next link in the chain.

ANSWER:

```
public class Link {
   private String word;
   private Link next;

   public Link(String w, Link n) {
      word = w;
      next=n;
   }
}
```

Linked List



- **String: word;**
- **Link: next;**

- **String: word;**
- **Link: next;**

- **String: word;**
- **Link: next;**

4b. **Write a recursive instance method that returns the word contained in the last link. (Hint: The last link's next reference is *null*):**

```
public String getLastValue() {
   if(next == null) // BASE CASE



      else



}
```

4c. **Write a recursive instance method that returns a reference to the last link:**

```
public Link getLastLink() {
   if(next == null) // BASE CASE



   else



}
```

4d. **Write a recursive instance method that returns the length of the list.**

4e. **A main method to create a list and display the last link:**

```
public static void main(String[] args) {
   Link head = new Link("One", new Link("Two", new Link("Three",
      new Link("Four", new Link("Five", new Link("Six", null))))));
   String lastValue = ___________________________________;
   System.out.println(lastValue);
}
```

4f. **Write a recursive instance method to print a string representation of the list:**

4g. **Write a recursive instance method to return a string with all the words concatenated together:**