Recursion

A method calls itself.

Look at it as if it calls another method with the same name and the same signature.
Recursion

Every recursion should have the following characteristics.

1. A simple **base case** which we have a solution for and a return value.
2. A way of getting our problem closer to the base case. I.e. a way to chop out part of the problem to get a somewhat simpler problem.
3. A **recursive call** which passes the simpler problem back into the method.
Recursion

Example: sum

Calculate the sum of the first $n$ positive integer numbers recursively.

```
int sum(int n)
{
    if(n==1)
        return 1;
    else
        return n + sum(n-1);
}
```
Write a recursive method which receives a string as parameter and count how many words are in the string, the word separator is the space character.

Solution:

```java
static public int countWords(String x){
    x = x.trim();
    if(x.indexOf(" ")<0)
        return 1;
    return 1+countWords(x.substring(x.indexOf(" ")+1));
}
```
Write a recursive method which receives an string s and prints the original string and then the next line without the last character and so on.

i.e. removeLastChar("Manuel")

output:
Manuel
Manue
Manu
Man
Ma
M

Solution:

```java
static public void removeLastChar(String s){
    if(s.length() == 1)
        System.out.println(s);
    else{
        System.out.println(s);
        removeLastChar(s.substring(0, s.length()-1));
    }
}
```
Recursion - Problem 3

Write a recursive method which calculates the $p$ power of a number $n$.

Solution:

```java
static public int power(int n, int p){
    if(p==1)
        return n;
    return n * power(n, p-1);
}
```
Recursion - Problem 3 cont.

Does the previous solution handle power of zero??

\[ n^0 = 1 \]

How would you add support for this?

Does your previous solution handle negative powers??

\[ n^{-2} = 0.25 \]

How would you add support for this?
Solution:

```java
static public float power(int n, int p){
    if(p==0)
        return 1;
    if(p==1)
        return n;
    if(p>0)
        return n * power(n, p-1);
    else
        return (float)1/n * (power(n, p+1));
}
```
Write a recursive method which receives a sorted array of integers (myA), a value (n) to search and two extra values (min & max) which are the lowest and highest inclusive indices that are searched. It should return the index where the value was found or -1 if the value is not in the array.

```java
static public int search(int[] myA, int n, int min, int max) {
    int mid;
    if(min>max)
        return -1;
    else{
        mid = (min+max)/2;
        if(myA[mid]==n)
            return mid;
        else {
            if(myA[mid] < n)
                return search(myA, n, mid+1, max);
            else
                return search(myA, n, min, mid-1);
        }
    }
}
```