Recursion

Tail Recursion

Tail Recursion is when we have only one recursive call in the function and this is the last statement in the function:

```
public void countDown(int n)
{
    if (n > 0)
    {
        System.out.println(n);
        return countDown( n-1 );
    }
}
```

Is equivelent to:

```
public void countDown(int n)
{
    for (int x = n; x > 0; x--)
    {
        System.out.println(x);
    }
}
```

Non-Tail Recurison

Non-tail Recursion, as one would expect is when we have two or more recursive function calls:

```
public int Fib(n)
{
    return Fib(n-1) + Fib(n-2);
}
```

Indirect Recursion

Indirect Recursion is when two or more functions call eachother in a recursive fashion:

```java
public boolean isEven(int n)
{
    if ( n == 0)
    return true;
    if ( n > 0)
        return isOdd(n-1);
    else ( n < 0)
        return isOdd(n+1);
}

public boolean isOdd(int n)
{
    if ( n == 0)
        return false;
    if ( n > 0)
        return isEven(n-1);
    else ( n < 0)
        return isEven(n+1);
}
```

Nested Recursion

Nested recursion is when the parameter of a recursive call is another recursive call. As one can imaging these functions tend to explode in computational time and complexity.

Backtracking

This is the methology that if a recursive path does not find a solution, it will move back to find another path to follow until a solution is found or the list of paths is exhausted.

A good demonstration of this is the **8 Queens Problem**.

Excessive Recursion

Recursion does have its trade-offs. Common issues attached to recursion are:

1. Redundant/unintelligent function calls.
2. Too many stack frames = stack-overflow.

Regan Koopmans 2016.