Recursion

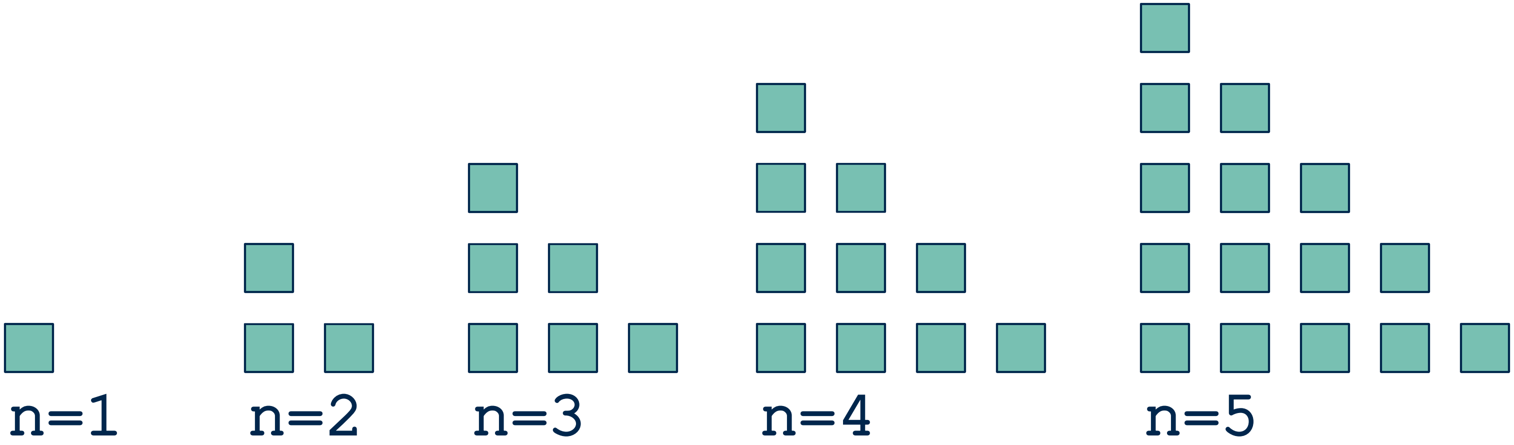
- A programming technique in which a function calls itself (repeatedly).

- It works!

- Loops are not the only way.
- Recursion is arguably a more “natural” way to program than imperative programming.

The Triangular Numbers

1, 3, 6, 10, 15, 21 …



The Way You Know

```
int tri(int n) {
  int sum = 0;
  for (int i=0; i<n; i++)
    sum += i;
  return sum;
}
```

How Do We Compute the `n`th Triangular Number?

- What is the relationship between the `n`th triangular number and the (`n-1`)th?
- The `n`th triangular number is the sum of `n` and the (`n-1`)th triangular number.
  - `T(n) = n + T(n-1)`
- When to stop?
  - `T(1) = 1`

The Recursive Way

```
int tri(int n) {
  if (n==1)
    return 1;
  return(n + tri(n-1));
}
```

What's Really Happening?

- Suppose we compute the 5th triangular number using our recursive function.

`tri(5)`expands to:

```
5+tri(4) ==> 5+4+tri(3) ==>
5+4+3+tri(2) ==> 5+4+3+3+tri(1)
==> 5+4+3+2+1 ==> 15
```

Characteristics of a Recursive Method

- Calls itself.
- When calling itself, it makes the call on a smaller part of the problem.
- There is a base case which is simple enough that no recursive call needs to be made to reach a solution.

Writing a Recursive Method

- FIRST: Identify a relationship between the `n`th and (`n-1`)th call of the method.
  - e.g., Factorial: fac(n) = n * fac(n-1)

```
public class RFac1 {
    public static void main(String[] args) {
      for (int i=0; i<args.length; i++)
          System.out.println(args[i] + "  " + (new 
Rfac1()).doFac(Integer.parseInt(args[i])));
    }
    public int doFac(int f) {
      return f * doFac(f-1);
    }
}
```

Writing Recursive Methods -- more

- Second: identify the base case
  - Factorial:
    - 1! = 1, if less then 1, UDF

```
public class RFac {
    public static void main(String[] args) {
      for (int i=0; i<args.length; i++)
          System.out.println(args[i] + "  " + (new RFac
()).doFac(Integer.parseInt(args[i])));
    }
    public int doFac(int f) {
      if (f==1) return 1;
      if (f<1) return 0;
      return f * doFac(f-1);
    }
}
```

Permutations

- cat
  - cat
  - cta
  - atc
  - act
  - tca
  - tac

Rule of Generation

- Keeping the leftmost letter untouched.
- Anagram the rightmost `n-1` letters – recurse on the rightmost `n-1` letters.
- Rotate the original word `1` letter to the left – what falls off the front comes back at the back.
- Repeat the above `n` times.
- Stop when only 1 letter remains

cats

	start	leftmost	rightmost `n-1`	result
0	cats	c	ats, ast, tsa, tas, sat, sta	cats, cast, ctsa, ctas, csat, csta
1	atsc	a	tsc, tcs, sct, stc, cts, cst	atsc, atcs, asct, astc, acts, acst
2	tsca	t	sca, sac, cas, csa, asc, acs	tsca, tsac, tcas, tcsa, tasc, tacs
3	scat	s	cat, cta, atc, act, tca, tac	scat, scta, satc, sact, stca, stac

Permuter

```
public class Anagram {
    public static void main(String[] args) {
         (new Anagram()).doAnagram(args[0], 0);
    }
      public void doAnagram (String word, int loc){
         if (loc == (word.length()-1)) {
                  System.out.println(word);
                  return;
               }
         for(int i=loc; i<word.length(); i++) {
               doAnagram(word, loc+1);
               word = rotate(word, loc);
         }}
    private String rotate(String w, int l) {
     if (l>0)
      return w.substring(0,l)+w.substring(w.length()-1)+w.substring(l,w.length()-1);
     else
      return w.substring(w.length()-1)+w.substring(l, w.length()-1[redacted]
    }}
```

Binary Search

```
public int binsearch (int key){
  int lower=0,upper=nElem-1,index;
  while(lower <= upper) {
    index = (lower+upper)/2;
    if (a[index]==key) return key;
    else if (a[index] < key)
      lower = index+1;
    else
      upper = index-1; }
  return -1;
}
```

Recursive Binary Search

```
public int binrec (int key, int lower,
  int upper) {
  int index = (lower+upper)/2;
  if (lower > upper) return -1;
  if (a[index]==key) return key;
  else if (a[index] < key)
    return binrec(key,(lower+upper)/2+1,
                  upper);
  else
    return binrec(key,lower,
                  (lower+upper)/2-1);
}
```