Recursion Day 1

- Announcements:
    - Exam review session Wednesday@4pm in cs3331
    - Sam's Saturday OH rescheduled...
- Reminders:
    - Exam Thursday
    - P5 due tomorrow @ 11:59pm
    - P6 out Thursday after the exam

- **Functions**
    - **What is a function?**
        - Takes input, gives output!
        - Output is "return"ed
        - Custom operations!
            - "square" --> square(n)
            - "cube" --> cube(n)
            - "power" --> power(n,k) // generalizable!!!
            - "sum from 1 to b" --> sum(b)
            - "sum from a to b" --> sum(a,b)
            - "factorial" --> fact(n)
    - **Calling a function**
        - How?
            - `int answer = power(2,5);`
            - `int result = 10 + fact(4);`
            - `int blah = sum(11,100) - sum(10,100)`
            - `System.out.println(sum(2,313));`
        - What happens?
            1. Makes a new stack frame
            2. Input
                - Read the value on the stack
                - Put into a new variable in the stack
            3. Run the code
                - Side effects?
                - Printing
            4. Output
                - where does the output go?
            5. Remove the stack frame
        - *Example: Calling functions on their own line*
            - `fact(n)` --> what happens? NOTHING
            - What if we add a print statement to `fact`?
                - int result = fact(4) + fact(3) --> what gets prints??
        - **Calling void functions?**
            - voidFunction(input);

            - NOT: System.out.println(voidFunction(2,313));

- **Recursion intro**
    - *Factorial!*
        - Task: compute n!
        - Naive formulation:
            - n = n * (n-1) * (n-2) * (n-3) * ... * 1
        - Recursive formulation:

          ```
          fact(n) =   {1, if n = 1
                      {n * fact(n), otherwise
          ```

        - **Code it like this:**

          ```
          public static int fact(int n){
              if (n==1){
                  return 1;
              } else{
                  return n * fact(n-1);
              }
          }
          ```

        - Infinity?
            - Wait, but this means we need to call a function from itself...
            - We talked about this, isn't this infinite? No!
        - Tracing stack frames:
            - When we call fact(4) (`int r = fact(4)`) what happens in memory?
            - Stack frames + trace --> each time, n goes down by 1
            - Base case n =1 means we stop
            - Then return the value back down the stack
        - Parts of a recursive function:
            - Base case --> value where we STOP
            - Recursive call(s) --> the same function, but with a SMALLER input
            - Return value --> calculate the answer using the result of the recursive call
        - "Leap of faith" --> if the recursive call works, then the main function works!
    - *Example: Sums*
        - Task: compute 1+2+3+4+5...+n using recursion (i.e. **no loops**)
        - Recursive formulation:

          ```
          sum(n) =    {1, if n = 1
                      {n + sum(n-1), otherwise
          ```

        - Break it down:

  - Base case? n=1
  - Recursive call? sum(n-1)
  - Return value? n + sum(n-1)
- Code:

```
public static int sum(int n){
    if (n == 1){
        return 1;
    }

    return n + sum(n-1);
}
```

- Trace the stack for sum(3)
- ???:
  - what if we use the wrong base case?
  - what if we call it with an input < 1?