An Introduction to Recursion

"To understand recursion, you must first understand recursion"

Recursion

- A recursive function is a function that is defined in terms of itself.
- Every recursive definition must have a base case that is not recursive.
  - The non-recursive nature of the base case allows us to then solve previous recursive steps.
- There can be more than one base case.

Thinking Recursively

- Operation: Advance Wally forward until it can't go forward anymore

BASE CASE (Operation not defined in terms of itself)

  - If Wally can't move forward one step, stop.
  - Otherwise, move Wally forward one step and advance Wally forward until it can't go forward anymore.

RECURSION (Operation defined in terms of itself)

Factorial

- n! = n * (n-1) * (n-2) * ... * 2 * 1 for n > 0
  = 1 for n = 0
- But, since (n-1)! = (n-1) * (n-2) * 2 * 1, we can use recursion to define the factorial function:
  n! = n * (n-1)! for n > 0
  = 1 for n = 0 (base case)
- Example:
  4! = 4*3! = 4*(3*2!) = 4*(3*(2*1!)) = 4*(3*(2*(1*0!)))
  = 4*(3*(2*(1*1))) = 4*(3*(2*1)) = 4*(3*2) = 4*6 = 24

Factorial in Java

```
public static int factorial(int n) {
  // Precondition: n >= 0
  int result;
  if (n == 0)
     result = 1;
  else
     result = n * factorial(n-1);
  return result;
 }
```

Fibonacci Numbers in Java

$fib_n = fib_{n-1} + fib_{n-2}$ for n >= 2
$fib_1 = 1$
$fib_0 = 1$

```
public static int fib(int n) {
  // Precondition: n >= 0
  if (n <= 1)
     return n;
  else
     return fib(n-1) + fib(n-2);
}
```

Greatest Common Divisor

```
public static int gcd(int m, int n) {
  // Precondition: m > 0, n > 0
  if (m % n == 0)
     return n;
  else
     return gcd(n, m % n);
}
```

Sum of 1 + 2 + ... + n

```
public static int sum(int n) {
  // Precondition: n >= 1
  if (n == 1)
     return 1;
  else
     return ________________________;
}
```

Assert

`assert` *boolean_condition* `;`

- If assertion checking is enabled and the condition is false, the program terminates with an `AssertionError`.
- You can use asserts to test any condition you believe is true to make sure it really is during runtime. This is used for debugging purposes.
- Once you are done testing, you can run the program with assertion checking disabled for the program to run faster.
- Enabling Assertion Checking:
  - In Eclipse: Open Run Dialog, select Arguments, enter `-ea` for VM Arguments and click Apply.

Sum of 1 + 2 + ... + n again

```
public static int sum(int n) {
  assert n >= 1;
  if (n == 1)
     return 1;
  else
     return ____________________;
}
```