Outline

- Recursion
  - Definition and Setup
  - Tracing
    - Pitfalls
  - Running Time
- Examples
- Recursive Data Structures
- Conclusion
Common Pitfalls of Writing Recursive Methods

- *harmonic*(n) computes $1 + 1/2 + 1/3 + ... + 1/n$

- Pitfall 1: Missing base case

```java
public static double harmonic(int n) {
    return harmonic(n-1) + 1.0/n;
}
```

- Pitfall 2: No guarantee of convergence
  - Leads to infinite recursion (and stack overflow)

```java
public static double harmonic(int n) {
    if (n == 1) return 1.0;
    return harmonic(n) + 1.0/n;
}
```
Find and Fix This Bug!

- We want this method to print 0, 1, 2, ..., n-1, n
  - Trace it work n = 3

```java
public static void ascendingPrint(int n) {
    int i = 0;
    if (i <= n) {
        System.out.println(i);
        i++;
        ascendingPrint(n-1);
    }
}
```
We want this method to print 0, 1, 2, ..., n-1, n

```java
public static void ascendingPrint(int n) {
    int i = 0;
    if (i <= n) {
        System.out.println(i);
        i++;
        ascendingPrint(n-1);
    }
}
```

Fixed version below:

```java
/* Prints 0, 1, 2, ..., n
* @param n Number to print up to
* @param i Always pass in 0
*/
public static void ascendingPrint(int n, int i) {
    if (i <= n) {
        System.out.println(i);
        i++;
        ascendingPrint(n, i);
    }
}
```
Find and Fix This Bug! (Final)

Rewritten:

- Why is this better than before?

```java
/* Prints 0, 1, 2, ..., n
 * @param n Number to print up to
 */
public static void ascendingPrint(int n) {
    ascendingPrintHelper(n, 0);
}
private static void ascendingPrintHelper(int n, int i) {
    if (i <= n) {
        System.out.println(i);
        i++;
        ascendingPrintHelper(n, i);
    }
}
```
Fix Another Bug!

- Want to return the number of odd numbers in the given list.
- Idea (this is sound):
  - Remove first element from the list
  - If it's odd, then add 1 to the number of odds in the remaining list
  - If it's even, then add 0 to the number of odds in the remaining list

```java
public static int numOdd(List<Integer> list) {

}
```
Fix Another Bug!

- Want to return the number of odd numbers in the given list.
- Idea (this is sound):
  - Remove first element from the list
  - If it's odd, then add 1 to the number of odds in the remaining list
  - If it's even, then add 0 to the number of odds in the remaining list

```java
public static int numOdd(List<Integer> list) {
    int val = list.remove(0);
    if (val % 2 == 1) {
        return 1 + numOdd(list);
    }
    else {
        return numOdd(list);
    }
}
```

- Trace with [1, 2, 3], and edge cases (e.g., list with 1 item, empty list)
Want to return the sum of the odd numbers in the given list.

- Trace with [1, 2, 3], and edge cases (e.g., list with 1 item, empty list)

```java
public static int numOdd(List<Integer> list) {
    if (list.size() == 0) {
        return 0;
    }
    int val = list.remove(0);
    if (val % 2 == 1) {
        return 1 + numOdd(list);
    } else {
        return numOdd(list);
    }
}
```

(Challenge: list becomes empty after you call this method. What if that's not ok?)