Announcements:

- Project #1 due next Thursday (13 Feb).
- Test #1 Tuesday, 18 Feb at 8PM.
- AWE 61A Party this Sunday (9 Feb) in the Woz, 1-3PM.
- Guerilla Sections this weekend (see Piazza).
- Self-assessment quiz will be released tonight, due Monday. Watch the website and Piazza.

A Simple Recursion

```
def cascade(n):
    if n < 10:
        print(n)
    else:
        print(n)
        cascade(n//10)
    return

# Example output
>>> cascade(123)
123
12
1
>>> cascade(12)
12
1
>>> cascade(1)
1
```

- Each frame connects to the global frame.
- Frames without "Return value" are still active.
- Each recursive call has its own n value.
- That’s how it works, but try not to think of it this way!
- Think recursively instead.

Classifying Recursions: Linear Recursions

- Here, each call of `cascade` contains one recursive call.
- When that call completes, still a print to go.
- So calls must remain pending.
- A linear recursive process: total work and space proportional to depth of calls.

```
def triang(n):
    print(n)
    if n < 10:
        triang(n-1)
    return

# Example output
>>> triang(123)
123
12
1
```

- Again, each call of `triang` contains one recursive call.
- So this is a type of linear recursive process.
- But there’s no more to do when that call completes (tail recursive)
- So in principle, calls need not remain pending.
- An iterative process: total work still proportional to depth of calls, but total space need not be.
- This kind is suitable for a loop.

Classifying Recursion: Tree Recursions

- Previously, we looked at a program for computing values in the Fibonacci sequence:

```
def fib(n):
    """The Nth Fibonacci number, N>=0."""
    assert n >= 0
    if n <= 1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

Here, each invocation of `fib` makes two calls: work is exponential in depth of calls: A tree-recursive process.

```
def partitions(n, k):
    The number of non-decreasing sequences of two or more positive integers between 1 and k that add up to n.
    return

# Example output
>>> partitions(6, 4)
9:
2 + 4 = 6
1 + 1 + 4 = 6
3 + 3 = 6
1 + 2 + 3 = 6
1 + 1 + 1 + 3 = 6
2 + 2 + 2 = 6
1 + 1 + 2 + 2 = 6
1 + 1 + 1 + 1 + 2 = 6
1 + 1 + 1 + 1 + 1 + 1 = 6
```

A Tree Recursion: Partitions

- `partitions(n, k)`: The number of non-decreasing sequences of two or more positive integers between 1 and k that add up to n.
- For example, `partitions(6, 4)` is 9:

```
2 + 4 = 6
1 + 1 + 4 = 6
3 + 3 = 6
1 + 2 + 3 = 6
1 + 1 + 1 + 3 = 6
2 + 2 + 2 = 6
1 + 1 + 2 + 2 = 6
1 + 1 + 1 + 1 + 2 = 6
1 + 1 + 1 + 1 + 1 + 1 = 6
```
This leads to the following program:

def partitions(n, k):
    """The number of ways of partitioning N items into partitions of size <= K."""
    if n == 0:
        return 1
    elif n < 0 or k <= 0:
        return 0
    else:
        with_k = partitions(n-k, k)
        without_k = partitions(n, k-1)
        return with_k + without_k