Recursion

(Why yes, I suppose I did.)

Last couple of days: abstract.

Today: code with code sauce.
A method defined...

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }

    long c = secret1(i-1);
    return i * c;
}
```
A method defined...

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}
```
A method defined...

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}
```

\[ N! = \]
...in terms of itself.

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}
```

\[ N! = N \cdot (N - 1)! \]

Not so new after all.
...in terms of itself.

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}
```

\[
1! = 1
\]

\[
N! = N \cdot (N - 1)!
\]

Not so new after all.
...in terms of itself.

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}

N! = N \cdot (N - 1)!
```
Another one

```c
long secret2(long i, long j) {
    if (j == 0) {
        return 1;
    }
    return i * secret2(i, j-1);
}
```
Another one

```c
long secret2(int i, long j) {
    if (j == 0) {
        return 1;
    }
    return i * secret2(i, j-1);
}
```

$$m^0 = 1$$

$$m^n = m \cdot (m^{n-1})$$
Another one

```c
long secret2(int i, long j) {
    if (j == 0) {
        return 1;
    }
    return i * secret2(i, j-1);
}
```

\[ m^n = m \cdot (m^{n-1}) \]
long secret3(int i, int[] values) {
    if (i == values.length) {
        return 0;
    }
    return values[i] + secret3(i+1, values);
}
A pattern emerges

```java
long secret1(long i) {
    if (i == 1) {
        return 1;
    }

    long c = secret1(i-1);
    return i * c;
}

long secret2(long i, long j) {
    if (j == 0) {
        return 1;
    }

    return i * secret2(i, j-1);
}

long secret3(int i, int[] values) {
    if (i == values.length) {
        return 0;
    }

    return values[i] + secret3(i+1, values);
}
```
A pattern emerges

long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}

long secret2(long i, long j) {
    if (j == 0) {
        return 1;
    }
    return i * secret2(i, j-1);
}

long secret3(int i, int[] values) {
    if (i == values.length) {
        return 0;
    }
    return values[i] + secret3(i+1, values);
}

if some stopping condition
    return a value
store the result of a recursive call
compute the answer using that value
return the answer
long secret1(long i) {
    if (i == 1) {
        return 1;
    }

    long c = secret1(i-1);
    return i * c;
}

**Base Case**
- if some stopping condition
- return a value
- store the result of a recursive call
- compute the answer using that value
- return the answer

**Recursive Step**
**Terminology**

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}
```

1. Figure out how your problem gets smaller

**Base Case**
- if some stopping condition
- return a value
- store the result of a *recursive call*
- compute the answer using that value
- return the answer

**Recursive Step**
- An integer gets smaller
- You move one step further
- through an array
- or
- You move one step along a list.

---

**Mac’s Patented Human Algorithm for Writing Recursive Algorithms**

1. Figure out how your problem gets smaller

Duke Computer Science
long secret1(long i) {
    if (i == 1) {
        return 1;
    }
    long c = secret1(i-1);
    return i * c;
}

1. Figure out how your problem gets smaller
2. What's the smallest that can get?

An integer gets smaller or
You move one step further through an array or
You move one step along a list.

Often 0, or 1, or an empty list.

Mac's Patented Human Algorithm for Writing Recursive Algorithms

if some stopping condition
return a value
store the result of a recursive call
compute the answer using that value
return the answer
An integer gets smaller

You move one step further through an array

or

You move one step along a list.

Or

often 0, or 1, or an empty list.

We grade on this. Also, demo coming up!

Duke Computer Science

Terminology

long secret1(long i) {
    if (i == 1) {
        return 1;
    } else {
        long c = secret1(i-1);
        return i * c;
    }
}

if some stopping condition
    return a value
store the result of a recursive call
compute the answer using that value
return the answer

Base Case

Recursive Step

1. Figure out how your problem gets smaller or

2. What’s the smallest that can get?

3. That’s your base case. Write it!

Mac’s Patented Human Algorithm for Writing Recursive Algorithms

Wednesday, September 26, 12
Terminology

```c
long secret1(long i) {
    if (i == 1) {
        return 1;
    }  
    long c = secret1(i-1);  
    return i * c;
}
```

1. Figure out how your problem gets smaller
   
   2. What’s the smallest that can get?
   
   3. That’s your base case. Write it!
   
   4. Compute the answer to the one-smaller problem.

Mac’s Patented Human Algorithm for Writing Recursive Algorithms

An integer gets smaller
   
   or
   
   You move one step further through an array
   
   or
   
   You move one step along a list.

Often 0, or 1, or an empty list.

We grade on this. Also, demo coming up!

Duke Computer Science
Mac’s Patented Human Algorithm for Writing Recursive Algorithms

1. Figure out how your problem gets smaller
2. What’s the smallest that can get?
3. That’s your base case. Write it!
4. Compute the answer to the one-smaller problem.
5. Compute the answer to the this-sized problem.

An integer gets smaller or
You move one step further through an array or
You move one step along a list.

Often 0, or 1, or an empty list.

We grade on this. Also, demo coming up!

Wednesday, September 26, 12
Demo time!
countAs

isPalindrome

http://codingbat.com/java/Recursion-1